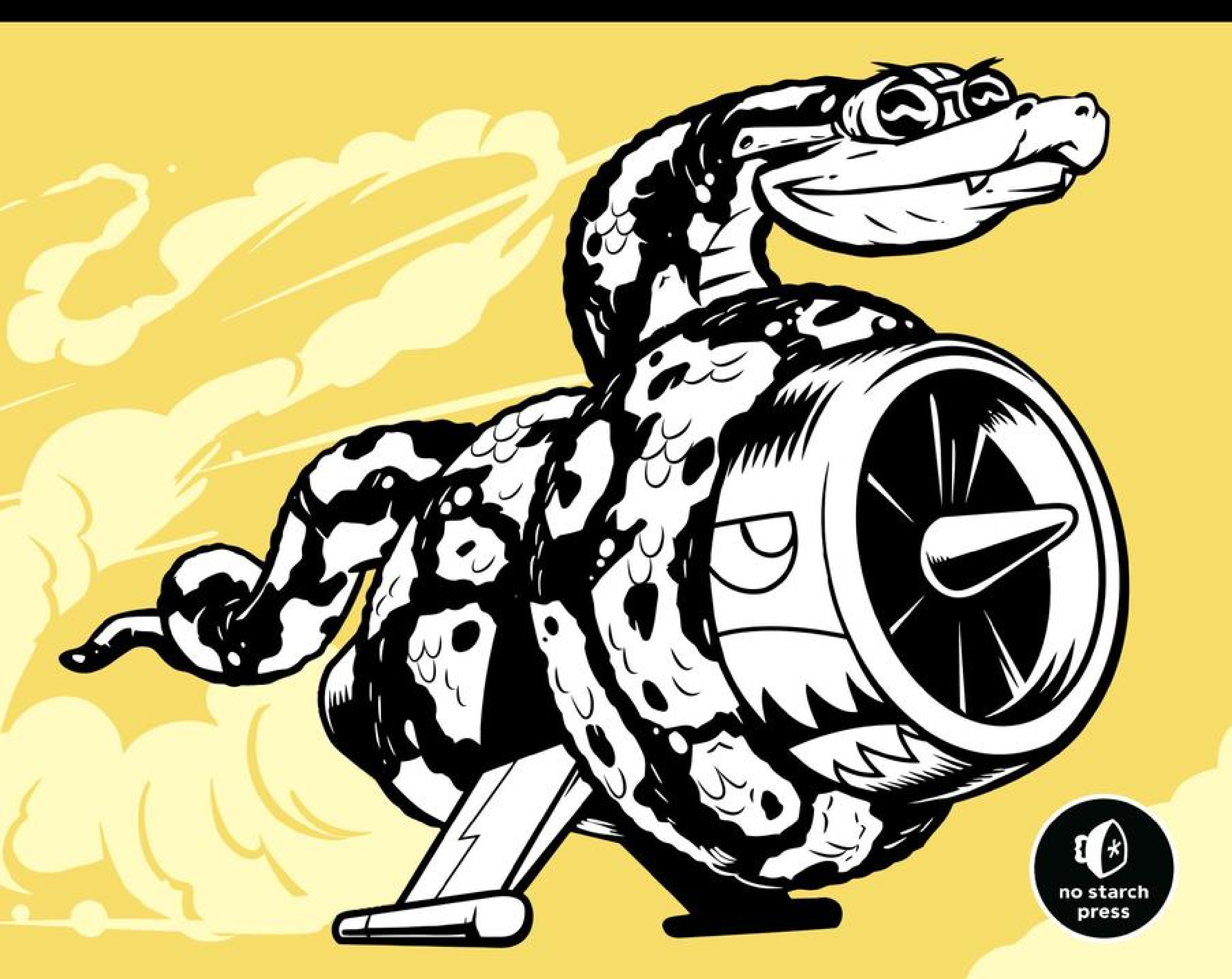
2ª EDICIÓN

PYTON CURSO CRASH

UNA PRÁCTICA , PROYECTO BASADO
INTRODUCCIÓN A LA PROGRAMACIÓN

ERIC MATTHES





ALABANZA POR Curso acelerado de Python

"Ha sido interesante ver a No Starch Press producir futuros clásicos que deberían estar junto a los libros de programación más tradicionales. *Python Crash Course* es uno de esos libros".
—Greg Laden, Blogs de ciencia

"Trata de algunos proyectos bastante complejos y los presenta de una manera coherente, lógica y agradable que atrae al lector hacia el tema".

—Revista de círculo completo

"Bien presentado con buenas explicaciones de los fragmentos de código. El libro trabaja con usted, un pequeño paso a la vez, construyendo un código más complejo, explicando lo que está pasando todo el camino".
—FlickThrough Reseñas

"¡Aprender Python con *Python Crash Course* fue una experiencia extremadamente positiva! Una gran elección si eres nuevo en Python."
—Mikke empieza a programar

"Hace lo que dice en la lata, y lo hace muy bien. . . . Presenta una gran cantidad de ejercicios útiles, así como tres proyectos desafiantes y entretenidos".
—RealPython.com

"Una introducción rápida pero completa a la programación con Python, *Python Crash Course* es otro libro excelente para agregar a su biblioteca y ayudarlo a finalmente dominar Python".
—TutorialEdge.net

"Una opción brillante para principiantes completos sin ninguna experiencia en codificación. Si está buscando una introducción sólida y sin complicaciones a este lenguaje tan profundo, tengo que recomendar este libro".
—WhatPixel.com

"Contiene literalmente todo lo que necesita saber sobre Python y aún más".

—FireBearStudio.com



pitón
Curso intensivo

2da edición

Una práctica , Proyecto basado
Introducción a la Programación

por Eric Mathes



San Francisco

Sobre el Autor

Eric Matthes es un profesor de ciencias y matemáticas de secundaria que vive en Alaska, donde imparte un curso de introducción a Python. Ha estado escribiendo programas desde que tenía cinco años. Eric actualmente se enfoca en escribir software que aborde las ineficiencias en la educación y traiga los beneficios del software de código abierto al campo de la educación. En su tiempo libre le gusta escalar montañas y pasar tiempo con su familia.

Acerca del revisor técnico

Kenneth Love ha sido programador, profesor y organizador de conferencias de Python durante muchos años. Ha hablado y enseñado en muchas conferencias, ha sido autónomo de Python y Django y actualmente es ingeniero de software para O'Reilly Media. Kenneth es co-creador del paquete django-braces, que proporciona varios mixins útiles para las vistas basadas en clases de Django. Puedes seguirlo en Twitter en @kennethlove.



Para mi padre, que siempre se hizo un
tiempo para responder a mis dudas sobre
programación, y para Ever, que apenas empieza
a hacerme sus preguntas.



Contenidos breves

Prefacio a la Segunda Edición. xvii

Agradecimientos xxi

Introducción xxxii

Parte I: Fundamentos. 1

Capítulo 1: Primeros pasos. 3

Capítulo 2: Variables y tipos de datos simples. 15

Capítulo 3: Introducción a las listas 33

Capítulo 4: Trabajar con listas.. 49

Capítulo 5: Sentencias if 71

Capítulo 6: Diccionarios. 91

Capítulo 7: Entrada de usuario y bucles while 113

Capítulo 8: Funciones 129

Capítulo 9: Clases. 157

Capítulo 10: Archivos y excepciones. 183

Capítulo 11: Prueba de su código 209

Parte II: Proyectos 223

Proyecto 1: Invasión alienígena

Capítulo 12: Un barco que dispara balas. 227

Capítulo 13: ¡Alienígenas! 255

Capítulo 14: Puntuación. 279

Proyecto 2: Visualización de datos

Capítulo 15: Generación de datos.. 305

Capítulo 16: Descarga de datos 333

Capítulo 17: Trabajo con API. 359

Proyecto 3: Aplicaciones Web

Capítulo 18: Primeros pasos con Django 379

Capítulo 19: Cuentas de usuario. 409

Capítulo 20: Estilo e implementación de una aplicación 437

Epílogo. 465

Apéndice A: Instalación y solución de problemas. 467

Apéndice B: Editores de texto e IDE. 473

Apéndice C: Obtención de ayuda 479

Apéndice D: Uso de Git para el control de versiones. 485

índice 495

Contenidos en detalle

Prefacio a la segunda edición xvii

Expresiones de gratitud xxi

Introducción xxxii

¿Para quién es este libro? . . . xiv
¿Qué puedes esperar aprender? . . . xiv
Recursos en línea . . . xxxv
¿Por qué Python? . . . xxxvi

Parte I: Fundamentos 1

1
Empezando 3

Configuración de su entorno de programación 3
Versiones de Python. 4
Ejecución de fragmentos de código Python. 4
Acerca del editor de texto sublime. 4
Python en diferentes sistemas operativos. . . . 5
Pitón en Windows. . . . 5
Pitón en macOS. . . . 7
Python es Linux. . . . 8
Ejecución de un programa Hola Mundo. . . . 9
Configuración de Sublime Text para usar la versión correcta de Python. . . . 9
Ejecutando hello_world.py . . . 10
Solución de problemas . . . 11
Ejecución de programas de Python desde una terminal. . . . 12
En Windows . . . 12
En macOS y Linux. . . . 12
Ejercicio 1-1: python.org. . . . 13
Ejercicio 1-2: Errores tipográficos de Hello World . . . 13
Ejercicio 1-3: Habilidades Infinitas. . . . 13
Resumen . . . 13

2
Variables y tipos de datos simples 15

Qué sucede realmente cuando ejecutas hello_world.py . . . 15
variables . . dieciséis
Nombrar y usar variables. . 17
Evitar errores de nombre al utilizar variables. . . 17
Las variables son etiquetas.................................... 18
Ejercicio 2-1: Mensaje Sencillo. . . . 19
Ejercicio 2-2: Mensajes simples. . . . 19

cuerdas . 19
Cambio de mayúsculas y minúsculas en una cadena con métodos. . . 20
Uso de variables en cadenas. 21
Adición de espacios en blanco a cadenas con tabulaciones o saltos de línea. . 22
Eliminación de espacios en blanco. . 22
Evitar errores de sintaxis con cadenas. . 24
Ejercicio 2-3: Mensaje personal. 25
Ejercicio 2-4: Casos de nombres 25
Ejercicio 2-5: Cita Famosa. 25
Ejercicio 2-6: Cita Famosa 2 25
Ejercicio 2-7: Eliminación de nombres. 25
numeros . . 25
enteros . 26
flotadores . . 26
Enteros y Flotantes. . . 27
Guiones bajos en Números. . . 28
Asignación Múltiple. . 28
constantes . . 28
Ejercicio 2-8: Número Ocho 29
Ejercicio 2-9: Número Favorito. 29
Comentarios 29
¿Cómo se escriben los comentarios? . . . 29
¿Qué tipo de comentarios debe escribir? . . 29
Ejercicio 2-10: Adición de comentarios. 30
El Zen de Python. . . 30
Ejercicio 2-11: Zen de Python. 31
Resumen 32

3 Introducción a las listas 33

¿Qué es una lista? 33
Acceso a los elementos de una lista. . . 34
Las posiciones de índice empiezan en 0, no en 1 35
Uso de valores individuales de una lista. . . 35
Ejercicio 3-1: Nombres 36
Ejercicio 3-2: Saludos. 36
Ejercicio 3-3: Su Propia Lista 36
Cambio, adición y eliminación de elementos. . . 36
Modificación de elementos en una lista. . . 36
Adición de elementos a una lista. . . 37
Eliminación de elementos de una lista. . . 38
Ejercicio 3-4: Lista de invitados 42
Ejercicio 3-5: Cambiar la lista de invitados. 42
Ejercicio 3-6: Más invitados 42
Ejercicio 3-7: Reducción de la lista de invitados. 43
Organización de una lista. 43
Ordenar una lista de forma permanente con el método sort(). . 43
Ordenar una lista temporalmente con la función sorted() . . 44
Imprimir una lista en orden inverso. . . 45
Encontrar la longitud de una lista. . 45
Ejercicio 3-8: Viendo el Mundo. 46
Ejercicio 3-9: Invitados a cenar 46
Ejercicio 3-10: Cada función. 46

Evitar errores de índice al trabajar con listas. 46
Ejercicio 3-11: Error intencional. 48
Resumen 48

4 Trabajar con listas que

49
recorren una lista completa. 49
Una mirada más cercana a los bucles. 50
Hacer más trabajo dentro de un bucle for. 51
Hacer algo después de un bucle for. 52
Evitar errores de sangría. . 53
Olvidarse de sangrar. . . 53
Olvidarse de aplicar sangría a las líneas adicionales. . 54
Sangría innecesariamente. . 55
Sangría innecesariamente después del bucle. . 55
Olvidando el Colón. . 56
Ejercicio 4-1: Pizzas. 56
Ejercicio 4-2: Animales 56
Realización de listas numéricas. . 57
Usando la función range(). . 57
Usando range() para hacer una lista de números. . 58
Estadísticas simples con una lista de números. . 59
Comprensiones de lista. . 59
Ejercicio 4-3: Contar hasta Veinte. 60
Ejercicio 4-4: Un Millón 60
Ejercicio 4-5: Sumar un millón. 60
Ejercicio 4-6: Números impares. 60
Ejercicio 4-7: Treses. 60
Ejercicio 4-8: Cubos 60
Ejercicio 4-9: Comprensión del Cubo. 60
Trabajar con parte de una lista. . 61
Cortar una lista. . 61
Bucle a través de una rebanada. . 62
Copiar una lista. . . 63
Ejercicio 4-10: Rebanadas . . . . sesenta y cinco
Ejercicio 4-11: Mis Pizzas, Tus Pizzas . . . . sesenta y cinco
Ejercicio 4-12: Más bucles. . . . . sesenta y cinco
tuplas. . . sesenta y cinco
Definición de una tupla. . 66
Recorriendo en bucle todos los valores de una tupla. . 67
Escribiendo sobre una Tupla. . 67
Ejercicio 4-13: Buffet. 68
Estilo de su código. 68
La guía de estilo. . 68
sangría . 69
Longitud de la línea . . 69
Líneas en blanco . . 69
Otras pautas de estilo. . . 70
Ejercicio 4-14: PEP 8 70
Ejercicio 4-15: Revisión de código 70
Resumen 70

5
si declaraciones 71

Un ejemplo sencillo. . . 72
Pruebas condicionales... 72
Comprobación de la igualdad. . . 72
Ignorar mayúsculas y minúsculas al comprobar la igualdad. . . 73
Comprobación de la desigualdad. . . 74
comparaciones numéricas. . . 74
Comprobación de múltiples condiciones. . . 75
Comprobar si un valor está en una lista. . . 76
Comprobar si un valor no está en una lista. . 77
Expresiones booleanas. . 77
Ejercicio 5-1: Pruebas condicionales. 78
Ejercicio 5-2: Más pruebas condicionales. 78
Si declaraciones. . . 78
Sentencias if simples. . 78
Declaraciones if-else. . 79
La cadena if-elif-else. . 80
Uso de múltiples bloques elif. . . 82
Omitiendo el bloque else. . 82
Prueba de múltiples condiciones. . 83
Ejercicio 5-3: Colores alienígenas #1. 84
Ejercicio 5-4: Alien Colors #2 84
Ejercicio 5-5: Colores alienígenas #3. 85
Ejercicio 5-6: Etapas de la Vida. 85
Ejercicio 5-7: Fruta favorita. 85
Uso de sentencias if con listas. . . 85
Comprobación de artículos especiales. . 86
Comprobación de que una lista no está vacía 87
Uso de listas múltiples. . . 88
Ejercicio 5-8: Hola administrador. 89
Ejercicio 5-9: Sin Usuarios 89
Ejercicio 5-10: Comprobación de nombres de usuario. 89
Ejercicio 5-11: Números ordinales. 89
Estilizando sus sentencias if. 90
Ejercicio 5-12: Dar estilo a declaraciones if 90
Ejercicio 5-13: Sus Ideas 90
Resumen 90

6
Diccionarios 91

Un diccionario sencillo. . 92
Trabajar con diccionarios. . 92
Acceso a valores en un diccionario. . 93
Adición de nuevos pares clave-valor. . 93
Comenzar con un diccionario vacío 94
Modificación de valores en un diccionario. . 95
Eliminación de pares clave-valor. . 96
Un diccionario de objetos similares. . . 97
Usando get() para acceder a los valores. . . 98

Ejercicio 6-1: Persona 99
Ejercicio 6-2: Números Favoritos 99
Ejercicio 6-3: Glosario 99
Bucle a través de un diccionario. . 99
Bucle a través de todos los pares clave-valor 99
Recorriendo todas las claves de un diccionario. . . 101
Bucle a través de las teclas de un diccionario en un orden particular. . . 103
Bucle a través de todos los valores en un diccionario. . . . 104
Ejercicio 6-4: Glosario 2 105
Ejercicio 6-5: Ríos. 105
Ejercicio 6-6: Sondeo 105
Anidamiento. . . 106
Una lista de diccionarios. . . 106
Una lista en un diccionario. . . 108
Un diccionario en un diccionario. . . . 110
Ejercicio 6-7: Personas 112
Ejercicio 6-8: Mascotas 112
Ejercicio 6-9: Lugares favoritos. 112
Ejercicio 6-10: Números Favoritos 112
Ejercicio 6-11: Ciudades 112
Ejercicio 6-12: Extensiones 112
Resumen 112

7
Entrada de usuario y bucles while 113

Cómo funciona la función input() . . 114
Escribir indicaciones claras. . 114
Uso de int() para aceptar entradas numéricas. . . 115
El Operador Módulo. . . . 116
Ejercicio 7-1: Coche de alquiler. 117
Ejercicio 7-2: Asientos en restaurante 117
Ejercicio 7-3: Múltiplos de Diez. 117
Introducción a los bucles while. 118
El ciclo while en acción. 118
Permitir que el usuario elija cuándo salir. 118
Uso de una bandera. 120
Usando break para salir de un bucle. 121
Uso de continuar en un bucle. 122
Evitar bucles infinitos. 122
Ejercicio 7-4: Ingredientes para pizza 123
Ejercicio 7-5: Boletos de cine. 123
Ejercicio 7-6: Tres Salidas. 124
Ejercicio 7-7: Infinito. 124
Uso de un bucle while con listas y diccionarios. 124
Mover elementos de una lista a otra. . 124
Eliminación de todas las instancias de valores específicos de una lista. . . 125
Llenar un diccionario con la entrada del usuario. . . . 126
Ejercicio 7-8: Charcutería 127
Ejercicio 7-9: Sin pastrami. 127
Ejercicio 7-10: Vacaciones de ensueño. 127
Resumen 127

8 Funciones 129

Definición de una función. . . 130
Pasar información a una función. . . 130
Argumentos y Parámetros. . . 131
Ejercicio 8-1: Mensaje . . . 131
Ejercicio 8-2: Libro Favorito. . . 131
Pasar Argumentos. . . . 131
Argumentos posicionales. . . 132
Argumentos de palabras clave. . . 133
Valores predeterminados . . . 134
Llamadas de funciones equivalentes 135
Evitar errores en los argumentos. . . 136
Ejercicio 8-3: Camiseta. . . 137
Ejercicio 8-4: Camisas Grandes. . . 137
Ejercicio 8-5: Ciudades . . . 137
Valores de retorno. . . 137
Devolviendo un valor simple. . . 138
Haciendo un Argumento Opcional. . . . 138
Devolver un diccionario. . . . 140
Uso de una función con un bucle while. . 141
Ejercicio 8-6: Nombres de ciudades . . . 142
Ejercicio 8-7: Álbum. . . 142
Ejercicio 8-8: Álbumes de usuario. . . 142
Pasar una lista. . . . 143
Modificación de una lista en una función. . . 143
Evitar que una función modifique una lista. . . . 145
Ejercicio 8-9: Mensajes . . . 146
Ejercicio 8-10: Envío de mensajes . . . 146
Ejercicio 8-11: Mensajes archivados. . . 146
Pasar un número arbitrario de argumentos. . . 147
Mezcla de argumentos posicionales y arbitrarios. . . . 148
Uso de argumentos arbitrarios de palabras clave. . . 148
Ejercicio 8-12: Sándwiches. . . 150
Ejercicio 8-13: Perfil de usuario. . . 150
Ejercicio 8-14: Coches . . . 150
Almacenamiento de sus funciones en módulos. . . 150
Importación de un módulo completo. . . 150
Importación de funciones específicas. . . 152
Uso de as para dar un alias a una función. . . 152
Uso de as para dar un alias a un módulo. . . 153
Importación de todas las funciones en un módulo. . . 153
Funciones de estilo. . . 154
Ejercicio 8-15: Modelos de impresión. . . 155
Ejercicio 8-16: Importaciones . . . 155
Ejercicio 8-17: Funciones de estilo . . . 155
Resumen . . . 155

9
Clases 157

Creación y uso de una clase. . . 158
Creando la Clase Perro. . . 158
Creación de una instancia de una clase. . . 160
Ejercicio 9-1: Restaurante 162
Ejercicio 9-2: Tres Restaurantes 162
Ejercicio 9-3: Usuarios 162
Trabajando con Clases e Instancias. . . 162
La clase de coche. . . 162
Establecer un valor predeterminado para un atributo. . . 163
Modificación de valores de atributos. . . 164
Ejercicio 9-4: Número servido 167
Ejercicio 9-5: intentos de inicio de sesión. 167
Herencia . . . 167
El método __init__() para una clase secundaria. . . 167
Definición de atributos y métodos para la clase secundaria. . . 169
Anulación de métodos de la clase principal. . . 170
Instancias como Atributos. . . 170
Modelado de objetos del mundo real. . . 173
Ejercicio 9-6: puesto de helados. 173
Ejercicio 9-7: Admin. 173
Ejercicio 9-8: Privilegios 173
Ejercicio 9-9: Actualización de la batería. 174
Importación de clases. . . 174
Importación de una sola clase. . . 174
Almacenamiento de varias clases en un módulo. . . 175
Importación de varias clases desde un módulo. . . 177
Importación de un módulo completo. . . 177
Importación de todas las clases de un módulo. . . 177
Importación de un módulo en un módulo. . . 178
Uso de alias. . . 179
Encontrar su propio flujo de trabajo. . . 179
Ejercicio 9-10: Restaurante importado. 180
Ejercicio 9-11: Administrador importado. 180
Ejercicio 9-12: Múltiples Módulos 180
La biblioteca estándar de Python. . . 180
Ejercicio 9-13: Dados. 181
Ejercicio 9-14: Lotería. 181
Ejercicio 9-15: Análisis de lotería. 181
Ejercicio 9-16: Módulo Python de la semana. 181
Clases de estilismo. 181
Resumen 182

10
Archivos y excepciones 183

Lectura de un archivo. 184
Lectura de un archivo completo. 184
Rutas de archivos. 185
Lectura línea por línea. 187
Elaboración de una lista de líneas a partir de un archivo. 188
Trabajar con el contenido de un archivo. 188

Archivos grandes: un millón de dígitos. . . . 189
¿Tu cumpleaños está contenido en Pi? . . . 190
Ejercicio 10-1: Aprendiendo Python. 191
Ejercicio 10-2: Aprendizaje C 191
Escribir en un archivo. . . . 191
Escribir en un archivo vacío. . . . 191
Escritura de varias líneas. . . . 192
Anexando a un archivo. . . . 193
Ejercicio 10-3: Invitado 193
Ejercicio 10-4: Libro de Visitas 193
Ejercicio 10-5: Encuesta de programación. 193
excepciones 194
Manejo de la excepción ZeroDivisionError . . . 194
Uso de bloques try-except. . . 194
Uso de excepciones para evitar bloqueos. . . 195
El otro bloque. . . 196
Manejo de la excepción FileNotFoundError . . . 197
Analizando Texto. . . . 198
Trabajando con Múltiples Archivos. . . . 199
Fallando en silencio. . . . 200
Decidir qué errores informar. . . 201
Ejercicio 10-6: Suma . . . 201
Ejercicio 10-7: Calculadora de Sumas 202
Ejercicio 10-8: Gatos y Perros 202
Ejercicio 10-9: Perros y gatos silenciosos 202
Ejercicio 10-10: Palabras Comunes. 202
Almacenamiento de datos 202
Usando json.dump() y json.load() . . . 203
Guardar y leer datos generados por el usuario. . . 204
Refactorización. . . 206
Ejercicio 10-11: Número favorito. 208
Ejercicio 10-12: Número favorito recordado. 208
Ejercicio 10-13: Verificar usuario. 208
Resumen 208

11

Probando tu código **209**

Prueba de una función. . . . 210
Pruebas unitarias y casos de prueba. . 211
Una prueba de aprobación. . 211
Una prueba fallida 212
Responder a una prueba fallida. . . . 213
Adición de nuevas pruebas. 214
Ejercicio 11-1: Ciudad, País 215
Ejercicio 11-2: Población 216
Prueba de una clase. . . . 216
Una variedad de métodos de afirmación. . . . 216
Una clase para probar... . . 217
Prueba de la clase AnonymousSurvey . . . 218
El método setUp(). . . . 220
Ejercicio 11-3: Empleado 221
Resumen . 222

Parte II: Proyectos 223

Proyecto 1: Invasión alienígena

12
Un barco que dispara balas 227

Planificación de su proyecto 228
Instalando Pygame. . . . 228
Comenzando el Proyecto del Juego. . . 229
Creación de una ventana de Pygame y respuesta a la entrada del usuario. . . . 229
Configuración del color de fondo. . . . 230
Creación de una clase de configuración. . . . 231
Adición de la imagen del barco. . . . 232
Creación de la clase de barco. . . . 233
Dibujando el Barco a la Pantalla. . . . 235
Refactorización: los métodos _check_events() y _update_screen() 236
El método _check_events() 236
El método _update_screen() . . . 237
Ejercicio 12-1: Cielo azul. 238
Ejercicio 12-2: Personaje del juego. 238
Pilotaje del barco. . . . 238
Respondiendo a una pulsación de tecla. . . . 238
Permitir el movimiento continuo. . . 239
Mover tanto a la izquierda como a la derecha. . . 240
Ajuste de la velocidad del barco. 241
Limitación del alcance del barco. . . . 243
Refactorización de _check_events() 243
Presionando Q para Salir. 244
Ejecutar el juego en modo de pantalla completa. . 244
Un resumen rápido. . . . 245
alien_invasion.py . . . 245
configuración.py. . . . 246
barco.py . . . 246
Ejercicio 12-3: Documentación de Pygame. 246
Ejercicio 12-4: Cohete 246
Ejercicio 12-5: Teclas 246
Disparar balas. . . . 246
Adición de la configuración de viñetas. . . . 247
Creación de la clase Bullet. . . . 247
Almacenamiento de balas en un grupo. . . 248
Disparando balas. . . . 249
Eliminación de viñetas antiguas. . . . 250
Limitación del número de balas. . . 251
Creando el método _update_bullets() 252
Ejercicio 12-6: tirador lateral. 253
Resumen . . . 253

13
¡Alienígenas! 255

Revisando el Proyecto. 256
Creando el primer alienígena. 256
Creando la clase alienígena. 257
Creando una Instancia del Alien 258
Construyendo la flota alienígena. 259
Determinar cuántos alienígenas caben en una fila. 260
Creación de una fila de alienígenas. 260
Refactorización de _create_fleet() 262
Adición de filas. 262
Ejercicio 13-1: Estrellas. 264
Ejercicio 13-2: Mejores Estrellas. 264
Haciendo el movimiento de la flota. 265
Moviendo a los alienígenas a la derecha. 265
Creación de ajustes para la dirección de la flota. 266
Comprobando si un alienígena ha tocado el borde. 266
Dejar caer la flota y cambiar de dirección. 267
Ejercicio 13-3: Gotas de lluvia. 268
Ejercicio 13-4: Lluvia constante. 268
Disparando a los extraterrestres. 268
Detección de colisiones de balas. 268
Fabricación de balas más grandes para pruebas. 270
Repoblación de la Flota. 270
Acelerando las balas. 271
Refactorización de _update_bullets() 271
Ejercicio 13-5: tirador lateral, parte 2 272
Finalización del juego 272
Detección de colisiones de naves y alienígenas. 272
Respuesta a colisiones de naves y extraterrestres. 273
Alienígenas que llegan al fondo de la pantalla. 276
¡Juego terminado! 276
Identificar cuándo deben ejecutarse partes del juego. 277
Ejercicio 13-6: Fin del juego. 278
Resumen 278

14
Puntuación 279

Adición del botón de reproducción. 280
Creación de una clase de botón. 280
Dibujar el botón en la pantalla. 281
Comenzando el Juego. 283
Restablecimiento del juego. 283
Desactivación del botón de reproducción. 284
Ocultar el cursor del ratón. 284
Ejercicio 14-1: Presione P para reproducir. 285
Ejercicio 14-2: Práctica de tiro 285
Subir de nivel 285
Modificación de la configuración de velocidad. 285
Restablecimiento de la velocidad. 287

Ejercicio 14-3: Práctica de tiro desafiante. 288
Ejercicio 14-4: Niveles de dificultad. 288
Puntuación 288
Visualización de la partitura 288
Realización de un marcador. . . . 289
Actualizando la partitura mientras los alienígenas son derribados. . . 291
Puesta a cero de la puntuación. . . 291
Asegurarse de anotar todos los golpes. . . 292
Aumento de los valores de los puntos. . . 292
Redondeando el marcador. . . . 293
Puntuaciones altas.. 294
Visualización del nivel. . . . 296
Visualización del número de barcos. . . 298
Ejercicio 14-5: Puntuación más alta de todos los tiempos. 301
Ejercicio 14-6: Refactorización. 301
Ejercicio 14-7: Expandiendo el Juego. 302
Ejercicio 14-8: tirador lateral, versión final. 302
Resumen 302

Proyecto 2: Visualización de datos

15 Generación de datos 305

Instalación de Matplotlib. . . 306
Trazar un gráfico de línea simple. . . . 306
Cambiar el tipo de etiqueta y el grosor de línea. . . 307
Corrección de la Trama. . . 309
Uso de estilos integrados. . . . 310
Trazado y estilo de puntos individuales con scatter() . . . 310
Trazar una Serie de Puntos con scatter() 312
Cálculo de datos automáticamente. . . . 312
Definición de colores personalizados. . . 314
Uso de un mapa de colores. . . 314
Guardar sus parcelas automáticamente. . . 315
Ejercicio 15-1: Cubos 315
Ejercicio 15-2: Cubos de colores. 315
Paseos aleatorios. . . . 315
Creando la Clase RandomWalk() . . . 316
Elegir direcciones. . . 316
Trazado del Paseo Aleatorio. . . 317
Generación de paseos aleatorios múltiples. . . 318
Estilizando el Paseo. . . . 319
Ejercicio 15-3: Movimiento Molecular. 323
Ejercicio 15-4: Paseos aleatorios modificados. 323
Ejercicio 15-5: Refactorización. 323
Tirar dados con Plotly. 323
Instalación de Plotly. 324
Creación de la clase de troquel. 324
Tirar el dado. 325

Análisis de los resultados. . . 325
Elaboración de un histograma 326
Tirar dos dados. . . 328
Dados rodantes de diferentes tamaños. . . 329
Ejercicio 15-6: Dos D8. 331
Ejercicio 15-7: Tres dados. 331
Ejercicio 15-8: Multiplicación. 331
Ejercicio 15-9: Comprensiones de dados. 331
Ejercicio 15-10: Practicando con Ambas Bibliotecas. 331
Resumen 331

dieciséis

Descarga de datos 333

El formato de archivo CSV. 334
Análisis de los encabezados del archivo CSV. . . 334
Impresión de los encabezados y sus posiciones. . . 335
Extracción y lectura de datos. . . 336
Trazado de datos en un gráfico de temperatura. . . 336
El módulo de fecha y hora. . . 337
Trazado de fechas. . . 338
Trazar un marco de tiempo más largo. . . 340
Trazado de una segunda serie de datos. . . 340
Sombreado de un área en el gráfico. . . 342
Comprobación de errores . . . 343
Descarga de sus propios datos. . . . 345
Ejercicio 16-1: Lluvia de Sitka. 346
Ejercicio 16-2: Comparación Sitka-Valle de la Muerte 346
Ejercicio 16-3: San Francisco. 346
Ejercicio 16-4: Índices automáticos. 347
Ejercicio 16-5: Explora 347
Asignación de conjuntos de datos globales: formato JSON. . . 347
Descarga de datos de terremotos. . . . 347
Examinando datos JSON. . . . 347
Hacer una lista de todos los terremotos. . . 350
Extrayendo Magnitudes. . . . 350
Extracción de datos de ubicación. . . . 351
Construcción de un mapa del mundo. . . 351
Una Manera Diferente de Especificar los Datos del Gráfico 353
Personalización del tamaño del marcador. . . 353
Personalización de los colores de los marcadores. . . 354
Otras escalas de colores. . . 356
Adición de texto flotante. . . 356
Ejercicio 16-6: Refactorización. 357
Ejercicio 16-7: Título automatizado. 357
Ejercicio 16-8: Terremotos Recientes 358
Ejercicio 16-9: Incendios Mundiales. 358
Resumen 358

17
Trabajar con API 359

Usando una API . . . 359
web Git y GitHub. . . . 360
Solicitud de datos mediante una llamada a la API. . . . 360
Solicitudes de instalación. . . . 361
Procesamiento de una respuesta API. . . . 361
Trabajando con el Diccionario de respuestas. . . . 362
Resumen de los repositorios principales. . . . 364
Supervisión de los límites de velocidad de la API. . . . 365
Visualización de repositorios usando Plotly. . . . 366
Refinación de gráficos plotly. . . . 368
Adición de información sobre herramientas personalizada. . . . 369
Agregar enlaces en los que se puede hacer clic a nuestro gráfico. . . . 370
Más sobre Plotly y la API de GitHub . . . 371
La API de noticias de hackers . . . 372
Ejercicio 17-1: Otros idiomas. . . . 375
Ejercicio 17-2: Debates activos. . . . 375
Ejercicio 17-3: Probando python_repos.py . . . 375
Ejercicio 17-4: Exploración adicional . . . 375
Resumen . . . 375

Proyecto 3: Aplicaciones Web

18
Primeros pasos con Django 379

Configuración de un proyecto. . . . 380
Escribir una especificación. . . 380
Creación de un entorno virtual. . . . 380
Activación del Entorno Virtual. . . 381
Instalando Django. . . 381
Creación de un proyecto en Django. . . 382
Creación de la base de datos. . . 382
Visualización del Proyecto. . . 383
Ejercicio 18-1: Nuevos Proyectos. . . . 384
Inicio de una aplicación. . . 384
Definición de modelos. . . 385
Activación de modelos. . . 386
El sitio de administración de Django. . . 387
Definición del modelo de entrada. . . 390
Migración del modelo de entrada. . . 391
Registro de entrada con el sitio de administración. . . 391
El caparazón de Django. . . 392
Ejercicio 18-2: Entradas cortas. . . . 394
Ejercicio 18-3: La API de Django. . . . 394
Ejercicio 18-4: Pizzería. . . . 394
Creación de páginas: la página de inicio del registro de aprendizaje . . . 394
Mapeo de una URL. . . . 395
Escribir una vista. . . . 396

Escribir una plantilla. . . . 397
Ejercicio 18-5: Planificador de comidas 398
Ejercicio 18-6: Página de inicio de pizzería 398
Creación de páginas adicionales. . . 398
Herencia de plantilla. . . 398
La página de temas. . . 400
Páginas de temas individuales. . . . 403
Ejercicio 18-7: Documentación de plantilla. 406
Ejercicio 18-8: Pizzería Páginas 406
Resumen 407

19 Cuentas de usuario 409

Permitir a los usuarios ingresar datos. . . . 410
Adición de nuevos temas. . . 410
Adición de nuevas entradas. 414
Edición de entradas. . . 418
Ejercicio 19-1: Blog 421
Configuración de cuentas de usuario. 421
La aplicación de los usuarios. 421
La página de inicio de sesión. . 422
Saliendo de tu cuenta . . 424
La página de registro. . . . 426
Ejercicio 19-2: Cuentas de blog. 428
Permitir que los usuarios sean dueños de sus datos. . . . 428
Restricción de acceso con @login_required . . . 429
Conexión de datos a ciertos usuarios. . . 430
Restricción del acceso a los temas a los usuarios apropiados. . . . 433
Protección de los temas de un usuario. . . 434
Protegiendo la página edit_entry. . . 434
Asociación de nuevos temas con el usuario actual. . . . 435
Ejercicio 19-3: Refactorización 436
Ejercicio 19-4: Protección de new_entry 436
Ejercicio 19-5: Blog Protegido. 436
Resumen 436

20 Diseño e implementación de una aplicación 437

Registro de aprendizaje de estilo. . . 438
La aplicación django-bootstrap4. . . . 438
Uso de Bootstrap para diseñar el registro de aprendizaje. . . . 438
Modificando base.html. . . 439
Diseñar la página de inicio usando un Jumbotron. . . . 443
Estilo de la página de inicio de sesión. . 444
Dar estilo a la página de temas. . . . 445
Dar estilo a las entradas en la página del tema. . . . 446
Ejercicio 20-1: Otras Formas 447
Ejercicio 20-2: Blog con estilo. 447
Implementación del registro de aprendizaje. 448
Hacer una cuenta de Heroku. 448
Instalación de la CLI de Heroku. 448

Instalación de paquetes necesarios. . . 448
Creando un archivo requirements.txt. . . . 448
Especificación del tiempo de ejecución de Python. . . 449
Modificando settings.py para Heroku. . . 450
Elaboración de un Procfile para Iniciar Procesos. . . 450
Uso de Git para rastrear los archivos del proyecto. . . 450
Empujando a Heroku. . . 452
Configuración de la base de datos en Heroku. . . 454
Refinando el despliegue de Heroku. . . 454
Asegurar el proyecto en vivo. . . 456
Confirmar y empujar cambios. . . 457
Configuración de variables de entorno en Heroku. . . 458
Creación de páginas de error personalizadas. . . 458
Desarrollo en curso 461
La configuración SECRET_KEY . . . 461
Eliminación de un proyecto en Heroku. . . 461
Ejercicio 20-3: Blog en vivo. . . . 462
Ejercicio 20-4: Más 404 462
Ejercicio 20-5: Registro de aprendizaje ampliado. . . . 462
Resumen 463

Epílogo **465**

UN
Instalación y solución de problemas **467**

Pitón en Windows. . . . 467
Encontrar el intérprete de Python. . . . 467
Agregar Python a su variable de ruta. . . . 468
Reinstalando Python. . . . 469
Pitón en macOS. . . . 469
Instalación de Homebrew. . . . 469
Instalando Phyton. . . . 470
Python es Linux. . . . 470
Palabras clave de Python y funciones integradas. . . . 471
Palabras clave de Python. . . . 471
Funciones integradas de Python. . . . 471

B
Editores de texto e IDE **473**

Personalización de la configuración de Sublime Text. . . . 474
Conversión de tabulaciones en espacios. . . . 474
Configuración del indicador de longitud de línea. . . . 474
Sangría y eliminación de sangría de bloques de código. . . . 474
Comentando bloques de código. . . . 475
Guardar su configuración. . . . 475
Otras personalizaciones. . . . 475
Otros editores de texto e IDE. . . . 475
INACTIVO 475
Geany. . . . 476
Emacs y Vim. . . . 476

átomo . . . 476
Código de estudio visual. . . 476
PyCharm. . . 476
Cuadernos Jupyter. . . 477

C
Obteniendo ayuda 479

Primeros pasos... 479
Vuelve a intentarlo . . . 480
Tomar un descanso. . . 480
Consulte los recursos de este libro. . . 480
Búsqueda en línea. . . 481
Desbordamiento de pila . . . 481
La documentación oficial de Python. . . 481
Documentación de la Biblioteca Oficial. . . 482
r/aprenderpython. . . 482
Publicaciones de blog. . . 482
Internet Relay Chat . . . 482
Crear una cuenta de IRC. . . 482
Canales para unirse. . . 483
Cultura IRC. . . 483
Flojo . . . 483
discordia . . . 484

D
Uso de Git para el control de versiones 485

Instalando Git. . . 486
Instalación de Git en Windows. . . 486
Instalación de Git en macOS. . . 486
Instalación de Git en Linux. . . 486
Configurando Git. . . 486
Realización de un proyecto... 486
Ignorar archivos. . . 487
Inicializar un Repositorio. . . 487
Comprobación del estado. . . 487
Adición de archivos al repositorio. . . 488
Haciendo un compromiso. . . 488
Comprobación del registro. . . 489
El Segundo Compromiso. . . 489
Revertir un cambio ... 490
Comprobación de confirmaciones anteriores. . . 491
Borrando el Repositorio. . . 493

Índice 495

Prefacio
la segunda edición

La respuesta a la primera edición de *Python Crash Course* ha sido abrumadoramente positiva. Se han impreso más de 500.000 copias, incluidas traducciones en ocho idiomas. He recibido cartas y correos electrónicos de lectores de hasta 10 años, así como de jubilados que quieren aprender a programar en su tiempo libre.
Python Crash Course se está utilizando en escuelas intermedias y secundarias, y también en clases universitarias. Los estudiantes a los que se les asignan libros de texto más avanzados utilizan *Python Crash Course* como texto complementario para sus clases y consideran que es un complemento valioso. La gente lo está utilizando para mejorar sus habilidades en el trabajo y para empezar a trabajar en sus propios proyectos paralelos. En resumen, la gente está usando el libro para toda la gama de propósitos que esperaba que hicieran.

La oportunidad de escribir una segunda edición de *Python Crash Course* ha sido completamente agradable. Aunque Python es un lenguaje maduro, continúa evolucionando como lo hacen todos los lenguajes. Mi objetivo al revisar el libro era hacerlo más ágil y sencillo. Ya no hay ninguna razón para aprender Python 2, por lo que esta edición se enfoca solo en Python 3. Muchos paquetes de Python se han vuelto más fáciles de instalar, por lo que las instrucciones de configuración e instalación son más sencillas. Agregué algunos temas de los que me di cuenta que los lectores se beneficiarían y actualicé algunas secciones para reflejar formas nuevas y más simples de hacer las cosas en Python. También he aclarado algunas secciones donde ciertos detalles de la

lenguaje no fueron presentados con tanta precisión como podrían haber sido. Todos los proyectos se han actualizado por completo utilizando bibliotecas populares y bien mantenidas que puede usar con confianza para crear sus propios proyectos.

El siguiente es un resumen de los cambios específicos que se han realizado en la segunda edición:

- En el Capítulo 1, las instrucciones para instalar Python se han simplificado para los usuarios de todos los principales sistemas operativos. Ahora recomiendo el editor de texto Sublime Text, que es popular entre los programadores principiantes y profesionales y funciona bien en todos los sistemas operativos.
- El Capítulo 2 incluye una descripción más precisa de cómo se implementado en Python. Las variables se describen como *etiquetas* para valores, lo que conduce a una mejor comprensión de cómo se comportan las variables en Python. El libro ahora usa f-strings, introducido en Python 3.6. Esta es una forma mucho más simple de usar valores de variables en cadenas. El uso de guiones bajos para representar números grandes, como 1_000_000, también se introdujo en Python 3.6 y se incluye en esta edición. La asignación múltiple de variables se introdujo previamente en uno de los proyectos, y esa descripción se ha generalizado y trasladado al Capítulo 2 para el beneficio de todos los lectores.
  Finalmente, en este capítulo se incluye una convención clara para representar valores constantes en Python.
- En el Capítulo 6, presento el método get() para recuperar valores de un diccionario, que puede devolver un valor predeterminado si no existe una clave.
- El proyecto Alien Invasion (Capítulos 12-14) ahora se basa completamente en clases. El juego en sí es una clase, más que una serie de funciones.
  Esto simplifica enormemente la estructura general del juego, reduciendo enormemente el número de llamadas a funciones y parámetros requeridos. Los lectores familiarizados con la primera edición apreciarán la simplicidad que proporciona este nuevo enfoque basado en clases. Pygame ahora se puede instalar en una línea en todos los sistemas, y los lectores tienen la opción de ejecutar el juego en modo de pantalla completa o en modo de ventana.
- En los proyectos de visualización de datos, las instrucciones de instalación de Matplotlib son más sencillas para todos los sistemas operativos. Las visualizaciones con Matplotlib utilizan la función subplots() , que será más fácil de desarrollar a medida que aprenda a crear visualizaciones más complejas.
  El proyecto Rolling Dice en el Capítulo 15 utiliza Plotly, una biblioteca de visualización bien mantenida que presenta una sintaxis limpia y una salida hermosa y totalmente personalizable.
- En el Capítulo 16, el proyecto meteorológico se basa en datos de la NOAA, que deberían ser más estables durante los próximos años que el sitio utilizado en la primera edición. El proyecto de mapeo se enfoca en la actividad sísmica global; al final de este proyecto, tendrá una visualización impresionante que muestra los límites de las placas tectónicas de la Tierra a través de un enfoque en las ubicaciones de todos los terremotos durante un período de tiempo determinado. Aprenderá a trazar cualquier conjunto de datos que involucre puntos geográficos.
- El Capítulo 17 utiliza Plotly para visualizar la actividad relacionada con Python en proyectos de código abierto en GitHub.

- El proyecto de Registro de aprendizaje (Capítulos 18ÿ20) está construido usando la última versión de Django y diseñado con la última versión de Bootstrap. El proceso de implementación del proyecto en Heroku se ha simplificado utilizando el paquete django-heroku y utiliza variables de entorno en lugar de modificar los archivos *settings.py* . Este es un enfoque más simple y es más consistente con la forma en que los programadores profesionales implementan proyectos modernos de Django.

- El Apéndice A se actualizó por completo para recomendar las mejores prácticas actuales. ticios en la instalación de Python. El Apéndice B incluye instrucciones detalladas para configurar Sublime Text y breves descripciones de la mayoría de los principales editores de texto e IDE en uso actual. El Apéndice C dirige a los lectores a recursos en línea más nuevos y populares para obtener ayuda, y el Apéndice D continúa ofreciendo un mini curso intensivo sobre el uso de Git para el control de versiones.
- El índice se ha actualizado completamente para permitirle usar *Python Crash Course* como referencia para todos sus futuros proyectos de Python.

¡Gracias por leer el *Curso acelerado de Python!* Si tiene algún comentario o pregunta, no dude en ponerse en contacto.



Expresiones de gratitud

Este libro no hubiera sido posible sin el maravilloso y extremadamente profesional personal de No Starch Press. Bill Pollock me invitó a escribir un libro introductorio y agradezco profundamente esa oferta original.

Tyler Ortman me ayudó a dar forma a mi forma de pensar en las primeras etapas de la redacción. Los comentarios iniciales de Liz Chadwick y Leslie Shen sobre cada capítulo fueron invaluables, y Anne Marie Walker ayudó a aclarar muchas partes del libro. Riley Hoffman respondió todas las preguntas que tenía sobre el proceso de ensamblar un libro completo y pacientemente convirtió mi trabajo en un hermoso producto terminado.

Me gustaría agradecer a Kenneth Love, el revisor técnico de *Python Crash Course.* Conocí a Kenneth en PyCon un año y su entusiasmo por el lenguaje y la comunidad de Python ha sido una fuente constante de inspiración profesional desde entonces. Kenneth fue más allá de la simple verificación de hechos y revisó el libro con el objetivo de ayudar a los programadores principiantes a desarrollar una comprensión sólida del lenguaje Python y la programación en general. Dicho esto, cualquier inexactitud que quede es completamente mía.

Me gustaría agradecer a mi padre por introducirme a la programación a una edad temprana y por no tener miedo de que rompiera su equipo. Me gustaría agradecer a mi esposa, Erin, por apoyarme y animarme a escribir este libro, y me gustaría agradecer a mi hijo, Ever, cuya curiosidad me inspira todos los días.



Introducción

Todo programador tiene una historia sobre cómo aprendió a escribir su primer programa. Empecé a programar de niño cuando mi padre trabajaba para Digital Equipment Corporation, una de las empresas pioneras de la era informática moderna. Escribí mi primer programa en un kit de computadora que mi papá había ensamblado en nuestro sótano. La computadora consistía en nada más que una placa base desnuda conectada a un teclado sin estuche, y su monitor era un tubo de rayos catódicos desnudo. Mi programa inicial era un simple juego de adivinanzas de números, que se parecía a esto:

```
¡Estoy pensando en un número! Intenta adivinar el número en el que estoy pensando: 25
¡Demasiado baja! Adivina otra vez: 50
¡Demasiado alto! Adivina otra vez: 42
¡Eso es todo! ¿Te gustaría volver a jugar? (sí/no) no
¡Gracias por jugar!
```

Siempre recordaré lo satisfecho que me sentía al ver a mi familia jugar un juego que creé y que funcionó como yo pretendía.

Esa primera experiencia tuvo un impacto duradero. Hay verdadera satisfacción en construir algo con un propósito, algo que solucione un problema. El software que escribo ahora satisface una necesidad más importante que los esfuerzos de mi infancia, pero la sensación de satisfacción que obtengo al crear un programa que funciona sigue siendo prácticamente la misma.

¿Para quién es este libro?

El objetivo de este libro es ponerlo al día con Python lo más rápido posible para que pueda crear programas que funcionen (juegos, visualizaciones de datos y aplicaciones web) mientras desarrolla una base en programación que le servirá bien para el resto de su vida. su vida. *Python Crash Course* está escrito para personas de cualquier edad que nunca antes han programado en Python o que nunca han programado en absoluto. Este libro es para aquellos que quieren aprender los conceptos básicos de la programación rápidamente para poder concentrarse en proyectos interesantes, y para aquellos a quienes les gusta probar su comprensión de nuevos conceptos resolviendo problemas significativos. *Python Crash Course* también es perfecto para profesores de secundaria y preparatoria que desean ofrecer a sus alumnos una introducción a la programación basada en proyectos. Si está tomando una clase universitaria y desea una introducción a Python más amigable que el texto que se le asignó, este libro también podría facilitar su clase.

¿Qué puedes esperar aprender?

El propósito de este libro es convertirte en un buen programador en general y en un buen programador de Python en particular. Aprenderá eficientemente y adoptará buenos hábitos mientras le brindo una base sólida en conceptos generales de programación. Después de trabajar en *Python Crash Course,* debería estar listo para pasar a técnicas de Python más avanzadas, y su próximo lenguaje de programación será aún más fácil de comprender.

En la primera parte de este libro, aprenderá los conceptos básicos de programación que necesita conocer para escribir programas en Python. Estos conceptos son los mismos que aprendería al comenzar en casi cualquier lenguaje de programación. Aprenderá sobre diferentes tipos de datos y las formas en que puede almacenar datos en listas y diccionarios dentro de sus programas. Aprenderá a crear colecciones de datos y a trabajar con esas colecciones de manera eficiente. Aprenderá a usar bucles while e instrucciones if para probar ciertas condiciones, de modo que pueda ejecutar secciones específicas de código mientras esas condiciones sean verdaderas y ejecutar otras secciones cuando no lo sean, una técnica que lo ayuda enormemente a automatizar procesos.

Aprenderá a aceptar la entrada de los usuarios para hacer que sus programas sean interactivos y mantenerlos en ejecución mientras el usuario esté activo. Explorará cómo escribir funciones para hacer que partes de su programa sean reutilizables, de modo que solo tenga que escribir bloques de código que realicen ciertas acciones una vez y luego usar ese código tantas veces como desee. A continuación, extenderá

este concepto a un comportamiento más complicado con las clases, haciendo que los programas relativamente simples respondan a una variedad de situaciones. Aprenderá a escribir programas que manejen correctamente los errores comunes. Después de trabajar con cada uno de estos conceptos básicos, escribirá algunos programas breves que resuelven algunos problemas bien definidos. Finalmente, dará su primer paso hacia la programación intermedia aprendiendo cómo escribir pruebas para su código para que pueda desarrollar sus programas aún más sin preocuparse por la introducción de errores. Toda la información de la Parte I lo preparará para emprender proyectos más grandes y complejos.

En la Parte II, aplicará lo que aprendió en la Parte I a tres proyectos. Puede hacer cualquiera o todos estos proyectos en el orden que mejor le convenga. En el primer proyecto (Capítulos 12–14), creará un juego de disparos *al estilo de Space Invaders* llamado *Alien Invasion,* que consiste en niveles de dificultad creciente. Una vez que haya completado este proyecto, debería estar bien encaminado para poder desarrollar sus propios juegos 2D.

El segundo proyecto (capítulos 15 a 17) lo introduce a la visualización de datos. Los científicos de datos tienen como objetivo dar sentido a la gran cantidad de información disponible para ellos a través de una variedad de técnicas de visualización. Trabajará con conjuntos de datos que genera a través del código, conjuntos de datos que descarga de fuentes en línea y conjuntos de datos que sus programas descargan automáticamente.
Una vez que haya completado este proyecto, podrá escribir programas que examinen grandes conjuntos de datos y hagan representaciones visuales de esa información almacenada.

En el tercer proyecto (Capítulos 18–20), construirá una pequeña aplicación web llamada Registro de aprendizaje. Este proyecto le permite llevar un diario de ideas y conceptos que ha aprendido sobre un tema específico. Podrá mantener registros separados para diferentes temas y permitir que otros creen una cuenta y comiencen sus propios diarios. También aprenderá cómo implementar su proyecto para que cualquiera pueda acceder a él en línea desde cualquier lugar.

Recursos en línea

Puede encontrar todos los recursos complementarios para el libro en línea en *https:// nostarch.com/ pythoncrashcourse2e/* o *http:// ehmatthes.github.io/ pcc_2e/.* Estos recursos incluyen:

Instrucciones de **configuración** Estas instrucciones son idénticas a las que se encuentran en el libro, pero incluyen enlaces activos en los que puede hacer clic para todas las piezas diferentes. Si tiene problemas de configuración, consulte este recurso.

Actualizaciones Python, como todos los lenguajes, está en constante evolución. Mantengo un conjunto completo de actualizaciones, por lo que si algo no funciona, consulte aquí para ver si las instrucciones han cambiado.

Soluciones a los ejercicios Debe pasar bastante tiempo por su cuenta intentando los ejercicios de las secciones "Pruébelo usted mismo". Pero si está atascado y no puede progresar, las soluciones para la mayoría de los ejercicios están en línea.

Hojas de trucos También hay en línea un juego completo de hojas de trucos descargables para una referencia rápida a los conceptos principales.

¿Por qué Python?

Todos los años considero si continuar usando Python o pasar a un lenguaje diferente, tal vez uno que sea más nuevo en el mundo de la programación. Pero sigo enfocándome en Python por muchas razones. Python es un lenguaje increíblemente eficiente: sus programas harán más en menos líneas de código de lo que requerirían muchos otros lenguajes. La sintaxis de Python también lo ayudará a escribir código "limpio". Su código será fácil de leer, fácil de depurar y fácil de ampliar y desarrollar en comparación con otros lenguajes.

La gente usa Python para muchos propósitos: crear juegos, crear aplicaciones web, resolver problemas comerciales y desarrollar herramientas internas en todo tipo de empresas interesantes. Python también se usa mucho en los campos científicos para la investigación académica y el trabajo aplicado.

Una de las razones más importantes por las que sigo usando Python es por la comunidad de Python, que incluye un grupo de personas increíblemente diverso y acogedor. La comunidad es esencial para los programadores porque la programación no es una actividad solitaria. La mayoría de nosotros, incluso los programadores más experimentados, necesitamos pedir consejo a otros que ya hayan resuelto problemas similares. Tener una comunidad bien conectada y de apoyo es fundamental para ayudarlo a resolver problemas, y la comunidad de Python brinda un apoyo total a las personas como usted que están aprendiendo Python como su primer lenguaje de programación.

Python es un gran lenguaje para aprender, ¡así que comencemos!

Parte I

Lo esencial

La Parte I de este libro le enseña los conceptos básicos que necesitará para escribir programas en Python. Muchos de estos conceptos son comunes a todos los lenguajes de programación, por lo que te serán útiles a lo largo de tu vida como programador.

En el **Capítulo 1 ,** instalará Python en su computadora y ejecutará su primera programa, que imprime el mensaje *Hello world!* a la pantalla

En el **Capítulo 2** aprenderá a almacenar información en variables y trabajar con texto y valores numéricos.

Los capítulos 3 y **4** introducen las listas. Las listas pueden almacenar tanta información como desee en una variable, lo que le permite trabajar con esos datos de manera eficiente. Podrá trabajar con cientos, miles e incluso millones de valores en solo unas pocas líneas de código.

En el **Capítulo 5 ,** usará declaraciones if para escribir código que responda de una manera si ciertas condiciones son verdaderas y responda de una manera diferente si esas condiciones no son verdaderas.

El Capítulo 6 le muestra cómo usar los diccionarios de Python, que le permiten hacer conexiones entre diferentes piezas de información. Al igual que las listas, los diccionarios pueden contener toda la información que necesite almacenar.

En el **Capítulo 7 ,** aprenderá a aceptar las entradas de los usuarios para que sus programas sean interactivos. También aprenderá sobre los bucles while , que ejecutan bloques de código repetidamente siempre que se cumplan ciertas condiciones.

En el **Capítulo 8 ,** escribirá funciones, que se denominan bloques de código que realizan una tarea específica y se pueden ejecutar siempre que los necesite.

El Capítulo 9 presenta clases, que le permiten modelar objetos del mundo real, como perros, gatos, personas, automóviles, cohetes y mucho más, para que su código pueda representar cualquier cosa real o abstracta.

El Capítulo 10 le muestra cómo trabajar con archivos y manejar errores para que sus programas no se bloqueen inesperadamente. Almacenará los datos antes de que se cierre el programa y volverá a leer los datos cuando el programa se ejecute de nuevo. Aprenderá sobre las excepciones de Python, que le permiten anticipar errores y hacer que sus programas manejen esos errores con gracia.

En el **Capítulo 11** aprenderá a escribir pruebas para su código para verificar que sus programas funcionen de la manera que desea. Como resultado, podrá expandir sus programas sin preocuparse por introducir nuevos errores.
Probar tu código es una de las primeras habilidades que te ayudarán en la transición de programador principiante a intermedio.

1

Empezando

En este capítulo, ejecutará su primer programa Python, *hello_world.py.* Primero, deberá verificar si una versión reciente de Python está instalado en su computadora; si no es así, lo instalará. También instalará un editor de texto para trabajar con sus programas Python. Los editores de texto reconocen el código de Python y resaltan secciones a medida que escribe, lo que facilita la comprensión de la estructura de su código.

Configuración de su entorno de programación

Python difiere ligeramente en diferentes sistemas operativos, por lo que deberá tener en cuenta algunas consideraciones. En las siguientes secciones, nos aseguraremos de que Python esté configurado correctamente en su sistema.

Versiones de Python

Cada lenguaje de programación evoluciona a medida que surgen nuevas ideas y tecnologías, y los desarrolladores de Python continuamente han hecho que el lenguaje sea más versátil y poderoso. Al momento de escribir este artículo, la última versión es Python 3.7, pero todo en este libro debe ejecutarse en Python 3.6 o posterior. En esta sección, averiguaremos si Python ya está instalado en su sistema y si necesita instalar una versión más nueva. El Apéndice A también contiene una guía completa para instalar la última versión de Python en cada uno de los principales sistemas operativos.

Algunos proyectos antiguos de Python todavía usan Python 2, pero debe usar Python 3. Si Python 2 está instalado en su sistema, probablemente esté allí para admitir algunos programas más antiguos que su sistema necesita. Dejaremos esta instalación como está y nos aseguraremos de que tenga una versión más reciente con la que trabajar.

Ejecución de fragmentos de código Python

Puede ejecutar el intérprete de Python en una ventana de terminal, lo que le permite probar fragmentos de código de Python sin tener que guardar y ejecutar un programa completo.

A lo largo de este libro, verá fragmentos de código que se ven así:

```
ÿ >>> print("¡Hola, intérprete de Python!")
¡Hola intérprete de Python!
```

El indicador >>> indica que debe usar la ventana de terminal, y el texto en negrita es el código que debe escribir y luego ejecutar presionando Intro. La mayoría de los ejemplos en el libro son programas pequeños e independientes que ejecutará desde su editor de texto en lugar de la terminal, porque escribirá la mayor parte de su código en el editor de texto. Pero a veces los conceptos básicos se mostrarán en una serie de fragmentos ejecutados a través de una sesión de terminal de Python para demostrar conceptos particulares de manera más eficiente. Cuando ve tres corchetes angulares en una lista de código ÿ, está viendo el código y la salida de una sesión de terminal. Intentaremos codificar el intérprete en su sistema en un momento.

¡También usaremos un editor de texto para crear un programa simple llamado *Hello World!* que se ha convertido en un elemento básico para aprender a programar. Hay una larga tradición en el mundo de la programación que imprimir un Hello world! mensaje a la pantalla ya que su primer programa en un nuevo idioma le traerá buena suerte. Un programa tan simple tiene un propósito muy real. Si se ejecuta correctamente en su sistema, cualquier programa de Python que escriba debería funcionar también.

Acerca del editor de texto sublime

Sublime Text es un editor de texto simple que se puede instalar en todos los sistemas operativos modernos. Sublime Text le permite ejecutar casi todos sus programas directamente desde el editor en lugar de a través de una terminal. Su código se ejecuta en una sesión de terminal incrustada en la ventana de Sublime Text, lo que facilita ver el resultado.

Sublime Text es un editor para principiantes, pero muchos programadores profesionales también lo usan. Si se siente cómodo usándolo mientras aprende Python, puede continuar usándolo a medida que avanza hacia proyectos más grandes y complicados. Sublime Text tiene una política de licencias muy liberal: puedes usar el editor de forma gratuita todo el tiempo que quieras, pero los desarrolladores te piden que compres una licencia si te gusta y quieres seguir usándola.

El Apéndice B proporciona información sobre otros editores de texto. Si tiene curiosidad acerca de las otras opciones, es posible que desee hojear ese apéndice en este punto. Si desea comenzar a programar rápidamente, puede usar Sublime Text para comenzar y considerar otros editores una vez que haya adquirido algo de experiencia como programador. En este capítulo, lo guiaré a través de la instalación de Sublime Text en su sistema operativo.

Python en diferentes sistemas operativos

Python es un lenguaje de programación multiplataforma, lo que significa que se ejecuta en todos los principales sistemas operativos. Cualquier programa de Python que escriba debe ejecutarse en cualquier computadora moderna que tenga Python instalado. Sin embargo, los métodos para configurar Python en diferentes sistemas operativos varían ligeramente.

En esta sección, aprenderá cómo configurar Python en su sistema. lo harás primero verifique si una versión reciente de Python está instalada en su sistema e instálela si no es así. Luego instalará Sublime Text. Estos son los únicos dos pasos que son diferentes para cada sistema operativo.

En las secciones siguientes, ejecutará *Hello World!* programar y solucionar cualquier problema que no haya funcionado. Lo guiaré a través de este proceso para cada sistema operativo, por lo que tendrá un entorno de programación de Python fácil de usar para principiantes.

Pitón en Windows

Windows no siempre viene con Python, por lo que probablemente deba instalarlo y luego instalar Sublime Text.

Instalación de Python

Primero, verifique si Python está instalado en su sistema. Abra una ventana de comandos ingresando **comando** en el menú Inicio o manteniendo presionada la tecla Mayús mientras hace clic con el botón derecho en su escritorio y seleccionando **Abrir ventana de comandos aquí** en el menú. En la ventana de la terminal, ingrese **python** en minúsculas. Si recibe un aviso de Python (>>>) en respuesta, Python está instalado en su sistema. Si ve un mensaje de error que le dice que python no es un comando reconocido, Python no está instalado.

En ese caso, o si ve una versión de Python anterior a Python 3.6, debe descargar un instalador de Python para Windows. Ir a *https:// python .org/* y coloque el cursor sobre el enlace **Descargas .** Debería ver un botón para descargar la última versión de Python. Haga clic en el botón, que debería comenzar a descargar automáticamente el instalador correcto para su sistema. Después

ha descargado el archivo, ejecute el instalador. Asegúrese de seleccionar la opción **Add Python to PATH,** lo que facilitará la configuración correcta de su sistema. La Figura 1-1 muestra esta opción seleccionada.

Figura 1-1: asegúrese de seleccionar la casilla de verificación etiquetada Add Python to PATH.

Ejecutar Python en una sesión de terminal

Abra una ventana de comando e ingrese **python** en minúsculas. Debería ver un indicador de Python (>>>), lo que significa que Windows ha encontrado la versión de Python que acaba de instalar.

```
C:\> pitón
Python 3.7.2 (v3.7.2:9a3ffc0492, 23 de diciembre de 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] en
win32
Escriba "ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

Si no ve este resultado o algo similar, consulte las instrucciones de configuración más detalladas. ciones en el Apéndice A.

Ingrese la siguiente línea en su sesión de Python y asegúrese de ver el intérprete de salida Hello Python!

```
>>> print("¡Hola, intérprete de Python!")
¡Hola intérprete de Python!
>>>
```

Cada vez que desee ejecutar un fragmento de código de Python, abra una ventana de comandos e inicie una sesión de terminal de Python. Para cerrar la sesión de terminal, presione ctrl-Z y luego presione enter, o ingrese el comando exit().

Instalación de texto sublime

Puede descargar un instalador para Sublime Text en *https:// sublimetext.com/.* Haga clic en el enlace de descarga y busque un instalador de Windows. Después de descargar el instalador, ejecútelo y acepte todos sus valores predeterminados.

Python en macOS

Python ya está instalado en la mayoría de los sistemas macOS, pero lo más probable es que sea una versión desactualizada que no querrá aprender. En esta sección, instalará la última versión de Python y luego instalará Sublime Text y se asegurará de que esté configurado correctamente.

Comprobando si Python 3 está instalado

Abra una ventana de terminal yendo a **Aplicaciones4Utilidades4Terminal.** También puede presionar la barra espaciadora z, escribir **terminal** y luego presionar Intro. Para ver qué versión de Python está instalada, ingrese **python** con una p minúscula ; esto también inicia el intérprete de Python dentro de la terminal, lo que le permite ingresar comandos de Python. Debería ver un resultado que le indica qué versión de Python está instalada en su sistema y un >>> aviso donde puede comenzar a ingresar comandos de Python, como este:

```
$ pitón
Python 2.7.15 (predeterminado, 17 de agosto de 2018, 22:39:05)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)] en darwin
Escriba "ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

Este resultado indica que Python 2.7.15 es actualmente la versión predeterminada instalado en esta computadora. Una vez que haya visto este resultado, presione ctrl-D o ingrese **exit()** para salir del indicador de Python y regresar al indicador de terminal.

Para verificar si tiene Python 3 instalado, ingrese el comando **pitón3.** Probablemente reciba un mensaje de error, lo que significa que no tiene instalada ninguna versión de Python 3. Si el resultado muestra que tiene instalado Python 3.6 o una versión posterior, puede pasar directamente a "Ejecución de Python en una sesión de terminal" en la página 8. Si Python 3 no está instalado de manera predeterminada, deberá instalarlo manualmente. Tenga en cuenta que cada vez que vea el comando python en este libro, debe usar el comando python3 en su lugar para asegurarse de que está usando Python 3, no Python 2; difieren lo suficiente como para que tenga problemas al intentar ejecutar el código de este libro con Python 2.

Si ve alguna versión anterior a Python 3.6, siga las instrucciones en la siguiente sección para instalar la última versión.

Instalación de la última versión de Python

Puede encontrar un instalador de Python para su sistema en *https:// python.org/.* Pase el cursor sobre el enlace **Descargar** y debería ver un botón para descargar la última versión de Python. Haga clic en el botón, que debería comenzar automáticamente

descargando el instalador correcto para su sistema. Después de que se descargue el archivo, ejecute el instalador.

Cuando haya terminado, ingrese lo siguiente en un indicador de terminal:

```
$ python3 --versión
Pitón 3.7.2
```

Debería ver un resultado similar a este, en cuyo caso, está listo para probar Python. Cada vez que vea el comando **python,** asegúrese de usar **python3.**

Ejecutar Python en una sesión de terminal

Ahora puede intentar ejecutar fragmentos de código de Python abriendo una terminal y escribiendo **python3.** Ingrese la siguiente línea en la sesión de terminal:

```
>>> print("¡Hola, intérprete de Python!")
¡Hola intérprete de Python!
>>>
```

Su mensaje debe imprimirse directamente en la ventana de terminal actual. Recuerda que puedes cerrar el intérprete de Python presionando ctrl-D o ingresando el comando exit().

Instalación de texto sublime

Para instalar el editor de Sublime Text, debe descargar el instalador en *https:// sublimetext.com/.* Haga clic en el enlace **Descargar** y busque un instalador para macOS. Después de que se descargue el instalador, ábralo y luego arrastre el ícono de Sublime Text a su carpeta de *Aplicaciones .*

pitón es linux

Los sistemas Linux están diseñados para la programación, por lo que Python ya está instalado en la mayoría de las computadoras con Linux. Las personas que escriben y mantienen Linux esperan que usted haga su propia programación en algún momento y lo alientan a hacerlo. Por esta razón, hay muy poco que instalar y solo unas pocas configuraciones que cambiar para comenzar a programar.

Comprobación de su versión de

Python Abra una ventana de terminal ejecutando la aplicación Terminal en su sistema (en Ubuntu, puede presionar ctrl-alt-T). Para averiguar qué versión de Python está instalada, ingrese **python3** con una p minúscula . Cuando se instala Python, este comando inicia el intérprete de Python. Debería ver una salida que indica qué versión de Python está instalada y un aviso >>> donde puede comenzar a ingresar comandos de Python, como este:

```
$ pitón3
Python 3.7.2 (predeterminado, 27 de diciembre de 2018, 04:01:51)
[CCG 7.3.0] en Linux
```

```
Escriba "ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

Este resultado indica que Python 3.7.2 es actualmente la versión predeterminada de Python instalada en esta computadora. Cuando haya visto este resultado, presione ctrl-D o ingrese **exit()** para salir del indicador de Python y regresar a un indicador de terminal. Cada vez que vea el comando **python** en este libro, ingrese **python3** en cambio.

Necesitará Python 3.6 o posterior para ejecutar el código de este libro. Si la versión de Python instalada en su sistema es anterior a Python 3.6, consulte el Apéndice A para instalar la última versión.

Ejecutar Python en una sesión de terminal

Puede intentar ejecutar fragmentos de código de Python abriendo una terminal e ingresando **python3,** como lo hizo al verificar su versión. Haga esto nuevamente, y cuando tenga Python ejecutándose, ingrese la siguiente línea en la sesión de terminal:

```
>>> print("¡Hola, intérprete de Python!")
¡Hola intérprete de Python!
>>>
```

El mensaje debe imprimirse directamente en la ventana de terminal actual. Recuerda que puedes cerrar el intérprete de Python presionando ctrl-D o ingresando el comando exit().

Instalación de texto sublime

En Linux, puede instalar Sublime Text desde el Centro de software de Ubuntu. Haga clic en el icono del software de Ubuntu en su menú y busque **Sublime Text.** Haga clic para instalarlo y luego ejecútelo.

Ejecución de un programa Hello World

Con una versión reciente de Python y Sublime Text instalada, está casi listo para ejecutar su primer programa de Python escrito en un editor de texto. Pero antes de hacerlo, debe asegurarse de que Sublime Text esté configurado para usar la versión correcta de Python en su sistema. Entonces escribirás el *Hello World!* programe y ejecútelo.

Configuración de Sublime Text para usar la versión correcta de Python

Si el comando python en su sistema ejecuta Python 3, no necesitará configurar nada y puede pasar a la siguiente sección. Si usas el python3 comando, deberá configurar Sublime Text para usar la versión correcta de Python cuando ejecute sus programas.

Haga clic en el icono de Sublime Text para iniciarlo, o busque Sublime Text en la barra de búsqueda de su sistema y luego inícielo. Vaya a **HerramientasÿSistema de compilación4 New Build System,** que abrirá un nuevo archivo de configuración para usted. Elimine lo que ve e ingrese lo siguiente:

Python3 .sublime-construir

```
{
    "cmd": ["python3", "-u", "$archivo"],
}
```

Este código le dice a Sublime Text que use el comando python3 de su sistema cuando ejecute sus archivos de programa de Python. Guarde el archivo como *Python3.sublime-build* en el directorio predeterminado que Sublime Text abre cuando elige Guardar.

Ejecutando hola_mundo.py

Antes de escribir su primer programa, cree una carpeta llamada *python_work* en algún lugar de su sistema para sus proyectos. Es mejor usar letras minúsculas y guiones bajos para los espacios en los nombres de archivos y carpetas, porque Python usa estas convenciones de nomenclatura.

Abra Sublime Text y guarde un archivo Python vacío **(File4Save As)** llamado *hello_world.py* en su carpeta *python_work .* La extensión .py le dice a Sublime Text que el código en su archivo está escrito en Python, lo que le indica cómo ejecutar el programa y resaltar el texto de una manera útil.

Una vez que haya guardado su archivo, ingrese la siguiente línea en el editor de texto:

hola_mundo.py

```
print("¡Hola, mundo de Python!")
```

Si el comando python funciona en su sistema, puede ejecutar su programa seleccionando **ToolsÿBuild** en el menú o presionando ctrl-B (zB en macOS). Si tuvo que configurar Sublime Text en la sección anterior, seleccione **ToolsÿBuild System** y luego seleccione **Python** 3. A partir de ahora podrá seleccionar **ToolsÿBuild** o simplemente presione ctrl-B (o zB) para ejecutar sus programas. .

Debería aparecer una pantalla de terminal en la parte inferior de la ganancia de Sublime Text dow, mostrando el siguiente resultado:

```
¡Hola mundo Python!
[Terminado en 0.1s]
```

Si no ve este resultado, es posible que algo haya fallado en el programa. Verifique cada carácter en la línea que ingresó. ¿Has escrito con mayúsculas por accidente? ¿Olvidaste una o ambas comillas o paréntesis? Los lenguajes de programación esperan una sintaxis muy específica y, si no la proporciona, obtendrá errores. Si no puede ejecutar el programa, consulte las sugerencias en la siguiente sección.

Solución de problemas

Si no puede ejecutar *hello_world.py* , aquí hay algunos remedios que puede probar que también son buenas soluciones generales para cualquier problema de programación:

- Cuando un programa contiene un error significativo, Python muestra un *rastro atrás,* que es un informe de error. Python revisa el archivo e intenta identificar el problema. Compruebe el rastreo; podría darle una pista sobre qué problema impide que se ejecute el programa.
- Aléjese de su computadora, tome un breve descanso y vuelva a intentarlo. Recuerde que la sintaxis es muy importante en la programación, por lo que incluso la falta de dos puntos, las comillas que no coinciden o los paréntesis que no coinciden pueden impedir que un programa se ejecute correctamente. Vuelva a leer las partes relevantes de este capítulo, revise su código e intente encontrar el error.
- Empezar de nuevo. Probablemente no necesite desinstalar ningún software, pero podría tener sentido eliminar su archivo *hello_world.py* y volver a crearlo desde cero.
- Pídale a otra persona que siga los pasos de este capítulo, en su computadora o en una diferente, y observe lo que hacen con cuidado. Es posible que te hayas saltado un pequeño paso que alguien más haya atrapado.
- Encuentre a alguien que conozca Python y pídale que lo ayude a configurarlo. Si pregunta, es posible que descubra que inesperadamente conoce a alguien que usa Python.
- Las instrucciones de configuración de este capítulo también están disponibles a través del sitio web complementario del libro en *https:// nostarch.com/ pythoncrashcourse2e/.* La versión en línea de estas instrucciones podría funcionar mejor para usted porque simplemente puede cortar y pegar el código.
- Pide ayuda en línea. El Apéndice C proporciona una serie de recursos, como foros y sitios de chat en vivo, donde puede solicitar soluciones a personas que ya han trabajado en el problema al que se enfrenta actualmente.

No te preocupes por molestar a los programadores experimentados. Todos los programadores se han quedado atascados en algún momento, y la mayoría de los programadores están felices de ayudarlo a configurar su sistema correctamente. Siempre que pueda indicar claramente lo que está tratando de hacer, lo que ya ha intentado y los resultados que está obteniendo, es muy probable que alguien pueda ayudarlo. Como se mencionó en la Introducción, la comunidad de Python es muy amigable y da la bienvenida a los principiantes.

Python debería funcionar bien en cualquier computadora moderna. Los problemas de configuración temprana pueden ser frustrantes, pero vale la pena solucionarlos. Una vez que recibes *hola _world.py* en ejecución, puede comenzar a aprender Python y su trabajo de programación se volverá más interesante y satisfactorio.

Ejecutar programas de Python desde una terminal

La mayoría de los programas que escriba en su editor de texto los ejecutará directamente desde el editor. Pero a veces es útil ejecutar programas desde una terminal. Por ejemplo, es posible que desee ejecutar un programa existente sin abrirlo para editarlo.

Puede hacer esto en cualquier sistema con Python instalado si sabe cómo acceder al directorio donde se almacena el archivo del programa. Para probar esto, asegúrese de haber guardado el archivo *hello_world.py* en la carpeta *python_work* de su escritorio.

en ventanas

Puede usar el comando de terminal cd, para *cambiar de directorio,* para navegar a través de su sistema de archivos en una ventana de comandos. El comando dir, para *directorio,* le muestra todos los archivos que existen en el directorio actual.

Abra una nueva ventana de terminal e ingrese los siguientes comandos para ejecutar *hello_world.py:*

```
ÿ C:\> cd Escritorio\python_work
ÿ C:\Escritorio\python_work> directorio
  hola_mundo.py
ÿ C:\Escritorio\python_work> python hello_world.py
  ¡Hola mundo Python!
```

En ÿ, usa el comando cd para navegar a la carpeta *python_work* , que se encuentra en la carpeta *Escritorio* . A continuación, utilice el comando dir para asegurarse de que *hello_world.py* esté en esta carpeta ÿ. Luego ejecuta el archivo usando el comando python hello_world.py ÿ.

La mayoría de sus programas funcionarán bien directamente desde su editor. Pero como su trabajo se vuelve más complejo, querrá ejecutar algunos de sus programas desde una terminal.

En macOS y Linux

Ejecutar un programa de Python desde una sesión de terminal es lo mismo en Linux y macOS. Puede usar el comando de terminal cd, para *cambiar de directorio,* para navegar a través de su sistema de archivos en una sesión de terminal. El comando ls, para *lista,* le muestra todos los archivos no ocultos que existen en el directorio actual.

Abra una nueva ventana de terminal e ingrese los siguientes comandos para ejecutar *hello_world.py:*

```
ÿ ~$ cd Escritorio/python_work/
ÿ ~/Escritorio/python_work$ ls
  hola_mundo.py
ÿ ~/Desktop/python_work$ python hello_world.py ¡Hola,
  mundo de Python!
```

En ÿ, usa el comando cd para navegar a la carpeta *python_work* , que se encuentra en la carpeta *Escritorio* . Luego, usa el comando ls para asegurarte de que *hello_world.py* esté en esta carpeta ÿ. Luego ejecuta el archivo usando el comando python hello_world.py ÿ.

Es así de simple. Simplemente use el comando python (o python3) para ejecutar programas de Python.

Inténtalo tú mismo

Los ejercicios de este capítulo son de naturaleza exploratoria. A partir del Capítulo 2, los desafíos que resolverás se basarán en lo que hayas aprendido.

1-1. python.org: explore la página de inicio de Python (https://python.org/) para encontrar temas que le interesen. A medida que se familiarice con Python, las diferentes partes del sitio le resultarán más útiles.

1-2. Errores tipográficos de **Hello World:** Abra el archivo hello_world.py que acaba de crear. Haga un error tipográfico en algún lugar de la línea y vuelva a ejecutar el programa. ¿Se puede hacer un error tipográfico que genera un error? ¿Puedes entender el mensaje de error? ¿Puedes hacer un error tipográfico que no genere un error? ¿Por qué crees que no hizo ¿un error?

1-3. Habilidades infinitas: si tuviera infinitas habilidades de programación, ¿qué construiría? Estás a punto de aprender a programar. Si tiene un objetivo final en mente, tendrá un uso inmediato para sus nuevas habilidades; ahora es un buen momento para redactar descripciones de lo que desea crear. Es un buen hábito mantener un cuaderno de "ideas" al que pueda consultar cada vez que desee comenzar un nuevo proyecto. Tómese unos minutos ahora para describir tres programas que desea crear.

Resumen

En este capítulo, aprendió un poco sobre Python en general e instaló Python en su sistema si aún no estaba allí. También instaló un editor de texto para facilitar la escritura de código Python. Ejecutó fragmentos de código de Python en una sesión de terminal y ejecutó su primer programa, *hello_world.py.* Probablemente también aprendiste un poco sobre la solución de problemas.

En el próximo capítulo, aprenderá sobre los diferentes tipos de datos con los que puede trabajar en sus programas de Python, y también usará variables.

2

V ariablesarena
Tipos de datos simples

En este capítulo aprenderá sobre los diferentes tipos de datos con los que puede trabajar en sus programas de Python. También aprenderá a usar variables para representar datos en su programas

Qué sucede realmente cuando ejecutas hello_world.py

Echemos un vistazo más de cerca a lo que hace Python cuando ejecuta *hello_world.py.* Resulta que Python hace una buena cantidad de trabajo, incluso cuando ejecuta un programa simple:

hola_mundo.py

```
print("¡Hola, mundo de Python!")
```

Cuando ejecute este código, debería ver este resultado:

```
¡Hola mundo Python!
```

Cuando ejecuta el archivo *hello_world.py,* la terminación .py indica que el archivo es un programa de Python. Luego, su editor ejecuta el archivo a través del *intérprete de Python,* que lee el programa y determina qué significa cada palabra en el programa. Por ejemplo, cuando el intérprete ve la palabra impresa seguida de paréntesis, imprime en la pantalla lo que está dentro de los paréntesis.

A medida que escribe sus programas, su editor resalta diferentes partes de su programa de diferentes maneras. Por ejemplo, reconoce que print() es el nombre de una función y muestra esa palabra en un color. Reconoce que "¡Hola, mundo de Python!" no es código de Python y muestra esa frase en un color diferente. Esta característica se llama *resaltado de sintaxis* y es bastante útil cuando comienza a escribir sus propios programas.

Variables

Intentemos usar una variable en *hello_world.py.* Agregue una nueva línea al principio del archivo y modifique la segunda línea:

hola_mundo.py

```
mensaje = "¡Hola, mundo de Python!"
imprimir (mensaje)
```

Ejecute este programa para ver qué sucede. Debería ver el mismo resultado que vio anteriormente:

```
¡Hola mundo Python!
```

Hemos agregado una *variable* llamada mensaje. Cada variable está conectada a un *value,* que es la información asociada a esa variable. En este caso, el valor es "Hello Python world!" texto.

Agregar una variable hace un poco más de trabajo para el intérprete de Python. Cuando procesa la primera línea, asocia el mensaje variable con "¡Hola, mundo de Python!" texto. Cuando llega a la segunda línea, imprime el valor asociado con el mensaje en la pantalla.

Ampliemos este programa modificando *hello_world.py* para imprimir un segundo mensaje. Agregue una línea en blanco a *hello_world.py* y luego agregue dos nuevas líneas de código:

```
mensaje = "¡Hola, mundo de Python!"
imprimir (mensaje)

mensaje = "¡Hola, mundo del Curso intensivo de Python!"
imprimir (mensaje)
```

Ahora, cuando ejecute *hello_world.py,* debería ver dos líneas de salida:

```
¡Hola mundo Python!
¡Hola, mundo del Curso intensivo de Python!
```

Puede cambiar el valor de una variable en su programa en cualquier momento, y Python siempre realizará un seguimiento de su valor actual.

Nombrar y usar variables

Cuando usa variables en Python, debe cumplir con algunas reglas y pautas. Romper algunas de estas reglas provocará errores; otras líneas de guía solo lo ayudan a escribir código que es más fácil de leer y comprender. Asegúrese de tener en cuenta las siguientes reglas variables:

- Los nombres de las variables solo pueden contener letras, números y guiones bajos. Pueden comenzar con una letra o un guión bajo, pero no con un número. Por ejemplo, puede llamar a una variable *mensaje_1* pero no a *1_mensaje.*
- No se permiten espacios en los nombres de las variables, pero se pueden usar guiones bajos para separar palabras en los nombres de las variables. Por ejemplo, *mensaje_saludo* funciona, pero *el mensaje de saludo* causará errores.
- Evite usar palabras clave de Python y nombres de funciones como nombres de variables; es decir, no use palabras que Python haya reservado para un propósito programático particular, como la palabra print. (Consulte “Palabras clave y funciones integradas de Python” en la página 471).
- Los nombres de las variables deben ser breves pero descriptivos. Por ejemplo, el *nombre* es mejor que n, *nombre_estudiante* es mejor que *s_n* y *longitud_nombre* es mejor que *longitud_nombre_de_personas.*
- Tenga cuidado al usar la letra minúscula *l* y la letra mayúscula *O* porque podrían confundirse con los números *1* y 0.

Puede tomar algo de práctica aprender a crear buenos nombres de variables, especialmente a medida que sus programas se vuelven más interesantes y complicados. A medida que escriba más programas y comience a leer el código de otras personas, mejorará en la creación de nombres significativos.

Las variables de Python que está utilizando en este punto deben estar en minúsculas. No obtendrá errores si usa letras mayúsculas, pero las letras mayúsculas en los nombres de variables tienen significados especiales que discutiremos en capítulos posteriores.

Evitar errores de nombre al usar variables

Todo programador comete errores, y la mayoría comete errores todos los días. Aunque los buenos programadores pueden crear errores, también saben cómo responder a esos errores de manera eficiente. Veamos un error que es probable que cometa al principio y aprendamos cómo solucionarlo.

Escribiremos algún código que genere un error a propósito. Introducir el siguiente código, incluido el mensaje de palabra mal escrito que *se* muestra en negrita:

```
mensaje = "¡Hola, lector del Curso acelerado de Python!"
imprimir (mensaje)
```

Cuando ocurre un error en su programa, el intérprete de Python hace todo lo posible para ayudarlo a descubrir dónde está el problema. El intérprete proporciona un rastreo cuando un programa no puede ejecutarse correctamente. Un *rastreo* es un registro de dónde el intérprete tuvo problemas al intentar ejecutar su código.
Este es un ejemplo del rastreo que proporciona Python después de haber escrito mal accidentalmente el nombre de una variable:

```
Rastreo (última llamada más reciente): ÿ Archivo
"hello_world.py", línea 2, en <módulo> ÿ imprimir (mensaje) ÿ
NameError: el nombre 'mensaje' no está definido
```

El resultado en ÿ informa que se produjo un error en la línea 2 del archivo *hello_world.py.* El intérprete muestra esta línea ÿ para ayudarnos a detectar el error rápidamente y nos dice qué tipo de error encontró ÿ. En este caso encontró un *error de nombre* e informa que la variable que se está imprimiendo, mensaje, no ha sido definida. Python no puede identificar el nombre de variable proporcionado. Un error de nombre generalmente significa que olvidamos establecer el valor de una variable antes de usarla o que cometimos un error de ortografía al ingresar el nombre de la variable.

Por supuesto, en este ejemplo omitimos la letra *s* en el mensaje de nombre de variable en la segunda línea. El intérprete de Python no revisa la ortografía de su código, pero se asegura de que los nombres de las variables se escriban de forma coherente. Por ejemplo, observe lo que sucede cuando también deletreamos *mensaje* incorrectamente en otro lugar del código:

```
mensaje = "¡Hola, lector del Curso acelerado de Python!"
imprimir (mensaje)
```

¡En este caso, el programa se ejecuta con éxito!

```
¡Hola, lector del Curso acelerado de Python!
```

Los lenguajes de programación son estrictos, pero ignoran la buena y la mala ortografía. Como resultado, no es necesario tener en cuenta las reglas gramaticales y ortográficas del inglés cuando intenta crear nombres de variables y escribir código.

Muchos errores de programación son simples errores tipográficos de un solo carácter en una línea de un programa. Si pasa mucho tiempo buscando uno de estos errores, sepa que está en buena compañía. Muchos programadores experimentados y talentosos pasan horas buscando este tipo de pequeños errores. Trate de reírse de eso y siga adelante, sabiendo que sucederá con frecuencia a lo largo de su vida como programador.

Las variables son etiquetas

Las variables a menudo se describen como cuadros en los que puede almacenar valores. Esta idea puede ser útil las primeras veces que use una variable, pero no es una forma precisa de describir cómo se representan internamente las variables en Python. Es mucho mejor pensar en las variables como etiquetas que puede asignar a los valores. También puede decir que una variable hace referencia a un determinado valor.

Esta distinción probablemente no importará mucho en sus programas iniciales, pero vale la pena aprender más temprano que tarde. En algún momento, verá un comportamiento inesperado de una variable y una comprensión precisa de cómo funcionan las variables lo ayudará a identificar lo que sucede en su código.

N ota *mejor manera de comprender los nuevos conceptos de programación es intentar usarlos en sus programas. Si te quedas atascado mientras trabajas en un ejercicio de este libro, intenta hacer otra cosa por un tiempo. Si todavía está atascado, revise la parte relevante de ese capítulo. Si aún necesita ayuda, consulte las sugerencias en el Apéndice C.*

Inténtalo tú mismo

Escriba un programa separado para realizar cada uno de estos ejercicios. Guarde cada programa con un nombre de archivo que siga las convenciones estándar de Python, usando letras minúsculas y guiones bajos, como simple_message.py y simple _mensajes.py.

2-1. Mensaje simple: asigne un mensaje a una variable y luego imprima ese mensaje.

2-2. Mensajes simples: asigne un mensaje a una variable e imprima ese mensaje. Luego cambie el valor de la variable a un nuevo mensaje e imprima el nuevo mensaje.

Instrumentos de cuerda

Debido a que la mayoría de los programas definen y recopilan algún tipo de datos, y luego hacen algo útil con ellos, es útil clasificar diferentes tipos de datos. El primer tipo de datos que veremos es la cadena. Las cadenas son bastante simples a primera vista, pero puede usarlas de muchas maneras diferentes.

Una *cadena* es una serie de caracteres. Cualquier cosa dentro de comillas se considera una cadena en Python, y puede usar comillas simples o dobles alrededor de sus cadenas de esta manera:

```
"Esto es una cadena".
'Esto también es una cadena.'
```

Esta flexibilidad le permite usar comillas y apóstrofes dentro de sus cadenas:

```
'Le dije a mi amigo: "¡Python es mi lenguaje favorito!"'
"El lenguaje 'Python' lleva el nombre de Monty Python, no de la serpiente".
"Una de las fortalezas de Python es su comunidad diversa y solidaria".
```

Exploremos algunas de las formas en que puede usar cadenas.

Cambio de mayúsculas y minúsculas en una cadena con métodos

Una de las tareas más simples que puede hacer con cadenas es cambiar el caso de las palabras en una cadena. Mire el siguiente código e intente determinar qué está sucediendo:

nombre.py

```
nombre = "ada lovelace"
imprimir (nombre. título ())
```

Guarde este archivo como *name.py* y luego ejecútelo. Debería ver esta salida:

```
ada lovelace
```

En este ejemplo, el nombre de la variable hace referencia a la cadena en minúsculas "ada lovelace". El método title() aparece después de la variable en la llamada print() . Un *método* es una acción que Python puede realizar en un dato. El punto (.) después del nombre en name.title() le dice a Python que haga que el método title() actúe sobre el nombre de la variable. Cada método va seguido de un conjunto de paréntesis, porque los métodos a menudo necesitan información adicional para hacer su trabajo. Esa información se proporciona entre paréntesis. La función title() no necesita información adicional, por lo que sus paréntesis están vacíos.

El método title() cambia cada palabra al caso del título, donde cada palabra comienza con una letra mayúscula. Esto es útil porque a menudo querrá pensar en un nombre como una pieza de información. Por ejemplo, es posible que desee que su programa reconozca los valores de entrada Ada, ADA y ada como el mismo nombre y los muestre todos como Ada.

Varios otros métodos útiles están disponibles para tratar el caso también. Por ejemplo, puede cambiar una cadena a letras mayúsculas o minúsculas como esta:

```
nombre = "Ada Lovelace"
imprimir (nombre. superior ())
imprimir (nombre. inferior ())
```

Esto mostrará lo siguiente:

```
ADA LOVELACE
ada lovelace
```

El método lower() es particularmente útil para almacenar datos. Muchas veces no querrá confiar en las mayúsculas que proporcionan sus usuarios, por lo que convertirá las cadenas a minúsculas antes de almacenarlas. Luego, cuando desee mostrar la información, utilizará el caso que tenga más sentido para cada cadena.

Uso de variables en cadenas

En algunas situaciones, querrá usar el valor de una variable dentro de una cadena. Por ejemplo, es posible que desee que dos variables representen un nombre y un apellido respectivamente, y luego desee combinar esos valores para mostrar el nombre completo de alguien:

nombre_completo.py

```
nombre = "ada"
last_name = "enlace de amor"
ÿ nombre_completo = f"{nombre} {apellido}"
imprimir (nombre_completo)
```

Para insertar el valor de una variable en una cadena, coloque la letra f inmediatamente antes de las comillas de apertura ÿ. Coloque llaves alrededor del nombre o nombres de cualquier variable que desee usar dentro de la cadena. Python reemplazará cada variable con su valor cuando se muestre la cadena.

Estas cadenas se llaman *cadenas f.* La *f* es para *formato,* porque Python formatea la cadena reemplazando el nombre de cualquier variable entre llaves con su valor. La salida del código anterior es:

```
ada lovelace
```

Puedes hacer mucho con f-strings. Por ejemplo, puede usar f-strings para redactar mensajes completos usando la información asociada con una variable, como se muestra aquí:

```
nombre = "ada"
last_name = "enlace de amor"
nombre_completo = f"{nombre} {apellido}"
ÿ print(f"Hola, {nombre_completo.título()}!")
```

El nombre completo se usa en una oración que saluda al usuario ÿ, y el método title() cambia el nombre a mayúsculas y minúsculas. Este código devuelve un saludo simple pero con un formato agradable:

```
¡Hola, Ada Lovelace!
```

También puede usar f-strings para redactar un mensaje y luego asignar el mensaje completo a una variable:

```
nombre = "ada"
last_name = "enlace de amor"
nombre_completo = f"{nombre} {apellido}"
ÿ mensaje = f"Hola, {nombre_completo.título()}!"
ÿ imprimir (mensaje)
```

Este código muestra el mensaje ¡Hola, Ada Lovelace! también, pero al asignar el mensaje a una variable ÿ hacemos que la llamada final a print() sea mucho más simple ÿ.

F-strings se introdujeron por primera vez en Python 3.6. Si está utilizando Python 3.5 o anterior, necesitará usar el método format() en lugar de esta *sintaxis f . Para usar format(), enumere las variables que desea usar en la cadena dentro de los paréntesis que siguen al formato. Se hace referencia a cada variable mediante un conjunto de llaves; las llaves se rellenarán con los valores enumerados entre paréntesis en el orden proporcionado:*

```
nombre_completo = "{} {}".formato(nombre, apellido)
```

Adición de espacios en blanco a cadenas con tabulaciones o saltos de línea

En programación, los espacios en *blanco* se refieren a cualquier carácter que no se imprima, como espacios, tabulaciones y símbolos de final de línea. Puede usar espacios en blanco para organizar su salida para que sea más fácil de leer para los usuarios.

Para agregar una tabulación a su texto, use la combinación de caracteres \t como se muestra en ÿ:

```
>>> imprimir("Pitón")
Python
ÿ >>> imprimir("\tPython")
        Pitón
```

Para agregar una nueva línea en una cadena, use la combinación de caracteres \n:

```
>>> print("Idiomas:\nPython\nC\nJavaScript")
Idiomas:
Pitón
C
JavaScript
```

También puede combinar tabulaciones y saltos de línea en una sola cadena. La cadena "\n\t" le dice a Python que se mueva a una nueva línea y comience la siguiente línea con una tabulación. El siguiente ejemplo muestra cómo puede usar una cadena de una línea para generar cuatro líneas de salida:

```
>>> print("Idiomas:\n\tPython\n\tC\n\tJavaScript")
Idiomas:
        Pitón
        C
        JavaScript
```

Las líneas nuevas y los tabuladores serán muy útiles en los próximos dos capítulos cuando comience a producir muchas líneas de salida a partir de unas pocas líneas de código.

Eliminación de espacios en blanco

Los espacios en blanco adicionales pueden ser confusos en sus programas. Para los programadores , 'python' y 'python' se parecen bastante. Pero para un programa, son dos cadenas diferentes. Python detecta el espacio adicional en 'python' y lo considera significativo a menos que le indique lo contrario.

Es importante pensar en los espacios en blanco, porque a menudo querrá comparar dos cadenas para determinar si son iguales. Por ejemplo, una instancia importante podría implicar verificar los nombres de usuario de las personas cuando inician sesión en un sitio web. Los espacios en blanco adicionales también pueden ser confusos en situaciones mucho más simples. Afortunadamente, Python facilita la eliminación de espacios en blanco superfluos de los datos que ingresan las personas.

Python puede buscar espacios en blanco adicionales en los lados derecho e izquierdo de una cadena. Para asegurarse de que no haya espacios en blanco en el extremo derecho de una cadena, utilice el método rstrip() .

```
ÿ >>> idioma_favorito = 'python' ÿ >>>
idioma_favorito ' python' ÿ >>>
    idioma_favorito.rstrip ()

    'pitón'
ÿ >>> idioma_favorito
    'pitón'
```

El valor asociado con favorite_language en ÿ contiene un espacio en blanco adicional al final de la cadena. Cuando le pide a Python este valor en una sesión de terminal, puede ver el espacio al final del valor ÿ. Cuando el método rstrip() actúa sobre la variable favorite_language en ÿ, se elimina este espacio extra. Sin embargo, solo se elimina temporalmente. Si vuelve a solicitar el valor de favorite_language , puede ver que la cadena se ve igual que cuando se ingresó, incluido el espacio en blanco adicional ÿ.

Para eliminar el espacio en blanco de la cadena de forma permanente, debe asociar el valor eliminado con el nombre de la variable:

```
    >>> idioma_favorito = 'python ' ÿ
>>> idioma_favorito = idioma_favorito.rstrip()
    >>> lenguaje_favorito
    'python'
```

Para quitar el espacio en blanco de la cadena, elimine el espacio en blanco del lado derecho de la cadena y luego asocie este nuevo valor con la variable original, como se muestra en ÿ. Cambiar el valor de una variable se hace a menudo en programación. Así es como se puede actualizar el valor de una variable a medida que se ejecuta un programa o en respuesta a la entrada del usuario.

También puede eliminar los espacios en blanco del lado izquierdo de una cadena usando el método lstrip() , o desde ambos lados a la vez usando strip():

```
                                   '           '
ÿ >>> idioma_favorito = python ÿ >>>
idioma_favorito.rstrip ()
    ' python'
ÿ >>> idioma_favorito.lstrip ()
    'python'
ÿ >>> idioma_favorito.strip ()
    'pitón'
```

En este ejemplo, comenzamos con un valor que tiene espacios en blanco al principio y al final ÿ. Luego eliminamos el espacio adicional del lado derecho en ÿ, del lado izquierdo en ÿ y de ambos lados en ÿ. Experimentar con estas funciones de eliminación puede ayudarlo a familiarizarse con la manipulación de cadenas. En el mundo real, estas funciones de eliminación se utilizan con mayor frecuencia para limpiar la entrada del usuario antes de que se almacene en un programa.

Evitar errores de sintaxis con cadenas

Un tipo de error que puede ver con cierta regularidad es un error de sintaxis. Se produce un *error de sintaxis* cuando Python no reconoce una sección de su programa como código de Python válido. Por ejemplo, si usa un apóstrofo entre comillas simples, generará un error. Esto sucede porque Python interpreta todo lo que se encuentra entre la primera comilla simple y el apóstrofe como una cadena. Luego intenta interpretar el resto del texto como código Python, que provoca errores.

Aquí se explica cómo usar comillas simples y dobles correctamente. Guarde este programa como *apostrophe.py* y luego ejecútelo:

apostrofe.py

```
mensaje = "Una de las fortalezas de Python es su comunidad diversa". imprimir
(mensaje)
```

El apóstrofe aparece dentro de un conjunto de comillas dobles, por lo que el intérprete de Python no tiene problemas para leer la cadena correctamente:

```
Una de las fortalezas de Python es su comunidad diversa.
```

Sin embargo, si usa comillas simples, Python no puede identificar dónde está la cadena. debería terminar:

```
mensaje = 'Una de las fortalezas de Python es su comunidad diversa.'
imprimir (mensaje)
```

Verá el siguiente resultado:

```
  Archivo "apostrophe.py", línea 1
    mensaje = 'Una de las fortalezas de Python es su comunidad diversa.'
                                  ^ÿ
Error de sintaxis: sintaxis invalida
```

En la salida puede ver que el error ocurre en ÿ justo después de la segunda comilla simple. Este error de sintaxis indica que el intérprete no reconoce algo en el código como código de Python válido. Los errores pueden provenir de una variedad de fuentes, y señalaré algunos comunes a medida que surjan. Es posible que vea errores de sintaxis a menudo a medida que aprende a escribir el código de Python adecuado. Los errores de sintaxis también son el tipo de error menos específico, por lo que pueden ser difíciles y frustrantes de identificar y corregir. Si se queda atascado en un error particularmente persistente, consulte las sugerencias en el Apéndice C.

N ota *función de resaltado de sintaxis de su editor debería ayudarlo a detectar algunos errores de sintaxis rápidamente mientras escribe sus programas. Si ve el código Python resaltado como si fuera inglés o inglés resaltado como si fuera código Python, es probable que tenga una comilla que no coincida en algún lugar de su archivo.*

Inténtalo tú mismo

Guarde cada uno de los siguientes ejercicios como un archivo separado con un nombre como nombre_casos.py. Si se atasca, tómese un descanso o consulte las sugerencias en el Apéndice C.

2-3. Mensaje personal: use una variable para representar el nombre de una persona e imprima un mensaje para esa persona. Su mensaje debe ser simple, como: "Hola Eric, ¿te gustaría aprender algo de Python hoy?"

2-4. Casos de nombres: use una variable para representar el nombre de una persona y luego imprima el nombre de esa persona en minúsculas, mayúsculas y mayúsculas y minúsculas.

2-5. Cita famosa: encuentra una cita de una persona famosa que admires. Imprima la cita y el nombre de su autor. Su salida debe ser similar a la siguiente, incluidas las comillas:

Albert Einstein dijo una vez: “Una persona que nunca cometió un error nunca intentó nada nuevo”.

2-6. Cita famosa 2: repite el ejercicio 2-5, pero esta vez, representa el nombre de la persona famosa usando una variable llamada persona_famosa. Luego, redacte su mensaje y represéntelo con una nueva variable llamada mensaje. Imprime tu mensaje.

2-7. Eliminación de nombres: use una variable para representar el nombre de una persona e incluya algunos caracteres de espacio en blanco al principio y al final del nombre. Asegúrese de usar cada combinación de caracteres, "\t" y "\n", al menos una vez.

Imprima el nombre una vez, de modo que se muestre el espacio en blanco alrededor del nombre. Luego imprima el nombre usando cada una de las tres funciones de eliminación, lstrip(), rstrip() y strip().

Números

Los números se utilizan con bastante frecuencia en la programación para llevar la puntuación en los juegos, representar datos en visualizaciones, almacenar información en aplicaciones web, etc. Python trata los números de varias maneras diferentes, dependiendo de cómo se usen. Primero veamos cómo Python maneja los números enteros, porque son los más simples para trabajar.

enteros

Puede sumar (+), restar (-), multiplicar (*) y dividir (/) enteros en Python.

```
>>> 2 + 3

5 >>> 3 - 2

1 >>> 2    3
* 6 >>>
3 / 2
1.5
```

En una sesión de terminal, Python simplemente devuelve el resultado de la operación. Python usa dos símbolos de multiplicación para representar exponentes:

```
>>> 3   ** 2

9 >>> 3 **   3

27 >>> 10 ** 6
1000000
```

Python también admite el orden de las operaciones, por lo que puede usar varias operaciones en una expresión. También puede usar paréntesis para modificar el orden de las operaciones para que Python pueda evaluar su expresión en el orden que especifique. Por ejemplo:

```
>>> 2 + 3*4
14
>>> (2 + 3) *        4
20
```

El espaciado en estos ejemplos no tiene efecto sobre cómo Python evalúa las expresiones; simplemente lo ayuda a detectar más rápidamente las operaciones que tienen prioridad cuando está leyendo el código.

flotadores

Python llama flotante a cualquier número con un punto decimal . Este término se usa en la mayoría de los lenguajes de programación y se refiere al hecho de que un punto decimal puede aparecer en cualquier posición de un número. Cada lenguaje de programación debe diseñarse cuidadosamente para administrar correctamente los números decimales, de modo que los números se comporten adecuadamente sin importar dónde aparezca el punto decimal.

En su mayor parte, puede usar decimales sin preocuparse por cómo se comportan. Simplemente ingrese los números que desea usar, y lo más probable es que Python haga lo que espera:

```
>>> 0,1 + 0,1
0.2
>>> 0,2 + 0,2
0,4
>>> 2 * 0,1
0,2
>>> 2 * 0,2
0.4
```

Pero tenga en cuenta que a veces puede obtener un número arbitrario de deci mal lugares en su respuesta:

```
>>> 0,2 + 0,1
0.30000000000000004 *
        0.1 >>> 3
0.30000000000000004
```

Esto sucede en todos los idiomas y es de poca importancia. Python intenta encontrar una manera de representar el resultado con la mayor precisión posible, lo que a veces es difícil dada la forma en que las computadoras tienen que representar números internamente. Simplemente ignore los lugares decimales adicionales por ahora; aprenderá formas de lidiar con los lugares adicionales cuando lo necesite en los proyectos de la Parte II.

Enteros y Flotantes

Cuando divide dos números, incluso si son números enteros que dan como resultado un número entero, siempre obtendrá un flotante:

```
>>> 4/2
2.0
```

Si mezcla un número entero y un flotante en cualquier otra operación, también obtendrá un flotante:

```
>>> 1 + 2,0
3.0
>>> 2* 6,0  3.0

>>> 3.0      ** 2
9.0
```

Python utiliza por defecto un flotante en cualquier operación que utilice un flotante, incluso si el la salida es un número entero.

Guiones bajos en números

Cuando escribe números largos, puede agrupar dígitos usando guiones bajos para que los números grandes sean más legibles:

```
>>> universo_edad = 14_000_000_000
```

Cuando imprime un número que se definió usando guiones bajos, Python imprime solo los dígitos:

```
>>> imprimir(universo_edad)
14000000000
```

Python ignora los guiones bajos al almacenar este tipo de valores. Incluso si no agrupa los dígitos de tres en tres, el valor no se verá afectado. Para Python, 1000 es lo mismo que 1_000, que es lo mismo que 10_00. Esta función funciona para números enteros y flotantes, pero solo está disponible en Python 3.6 y versiones posteriores.

Asignación Múltiple

Puede asignar valores a más de una variable usando una sola línea. Esto puede ayudar a acortar sus programas y hacerlos más fáciles de leer; utilizará esta técnica con mayor frecuencia al inicializar un conjunto de números.

Por ejemplo, así es como puede inicializar las variables x, y y z a cero:

```
>>> x, y, z = 0, 0, 0
```

Debe separar los nombres de las variables con comas y hacer lo mismo con los valores, y Python asignará cada valor a su variable posicionada respectivamente. Siempre que el número de valores coincida con el número de variables, Python los emparejará correctamente.

constantes

Una *constante* es como una variable cuyo valor permanece igual a lo largo de la vida de un programa. Python no tiene tipos constantes incorporados, pero los programadores de Python usan todas las letras mayúsculas para indicar que una variable debe tratarse como una constante y nunca cambiarse:

```
MAX_CONEXIONES = 5000
```

Cuando desee tratar una variable como una constante en su código, haga que el nombre de la variable esté en mayúsculas.

Inténtalo tú mismo

2-8. Número ocho: escriba operaciones de suma, resta, multiplicación y división que den como resultado el número 8. Asegúrese de encerrar sus operaciones en las llamadas print() para ver los resultados. Debes crear cuatro líneas que se vean así:

```
imprimir (5 + 3)
```

Su salida debería ser simplemente cuatro líneas con el número 8 apareciendo una vez en cada línea.

2-9. Número favorito: use una variable para representar su número favorito. Luego, usando esa variable, cree un mensaje que revele su número favorito. Imprime ese mensaje.

Comentarios

Los comentarios son una característica extremadamente útil en la mayoría de los lenguajes de programación. Todo lo que ha escrito en sus programas hasta ahora es código de Python. A medida que sus programas se vuelven más largos y complicados, debe agregar notas dentro de sus programas que describan su enfoque general del problema que está resolviendo. Un *comentario* le permite escribir notas en inglés dentro de sus programas.

¿Cómo se escriben los comentarios?

En Python, la marca hash (#) indica un comentario. El intérprete de Python ignora todo lo que sigue a una marca hash en su código. Por ejemplo:

comentario.py

```
# Di hola a todos.
print("¡Hola, gente de Python!")
```

Python ignora la primera línea y ejecuta la segunda línea.

```
¡Hola gente de Python!
```

¿Qué tipo de comentarios debe escribir?

La razón principal para escribir comentarios es explicar qué se supone que debe hacer su código y cómo lo está haciendo funcionar. Cuando estás en medio de un proyecto, entiendes cómo encajan todas las piezas. Pero cuando regresa a un proyecto después de un tiempo fuera, probablemente lo haya olvidado.

algunos de los detalles. Siempre puede estudiar su código por un tiempo y descubrir cómo se suponía que debían funcionar los segmentos, pero escribir buenos comentarios puede ahorrarle tiempo al resumir su enfoque general en un inglés claro.

Si desea convertirse en un programador profesional o colaborar con otros programadores, debe escribir comentarios significativos. Hoy en día, la mayor parte del software se escribe en colaboración, ya sea por un grupo de empleados de una empresa o por un grupo de personas que trabajan juntas en un proyecto de código abierto. Los programadores expertos esperan ver comentarios en el código, por lo que es mejor comenzar a agregar comentarios descriptivos a sus programas ahora. Escribir comentarios claros y concisos en su código es uno de los hábitos más beneficiosos que puede formar como nuevo programador.

Cuando esté determinando si escribir un comentario, pregúntese si tenías que considerar varios enfoques antes de encontrar una forma razonable de hacer que algo funcionara; si es así, escribe un comentario sobre tu solución. Es mucho más fácil eliminar comentarios adicionales más adelante que regresar y escribir comentarios para un programa escasamente comentado. De ahora en adelante, usaré comentarios en ejemplos a lo largo de este libro para ayudar a explicar secciones de código.

Inténtalo tú mismo

2-10. Agregar comentarios: elija dos de los programas que ha escrito y agregue al menos un comentario a cada uno. Si no tiene nada específico que escribir porque sus programas son demasiado simples en este punto, simplemente agregue su nombre y la fecha actual en la parte superior de cada archivo de programa. Luego escribe una oración que describa lo que hace el programa.

El Zen de Python

Los programadores experimentados de Python lo alentarán a evitar la complejidad y apuntar a la simplicidad siempre que sea posible. La filosofía de la comunidad de Python está contenida en “The Zen of Python” de Tim Peters. Puede acceder a este breve conjunto de principios para escribir un buen código de Python ingresando import this en su intérprete. No reproduciré todo el "Zen de Python" aquí, pero compartiré algunas líneas para ayudarlo a comprender por qué deberían ser importantes para usted como programador principiante de Python.

```
>>> import this El
zen de Python, de Tim Peters Lo bello es
mejor que lo feo.
```

Los programadores de Python aceptan la idea de que el código puede ser hermoso y elegante. En la programación, las personas resuelven problemas. Los programadores siempre han respetado las soluciones bien diseñadas, eficientes e incluso hermosas para

problemas. A medida que aprenda más sobre Python y lo use para escribir más código, alguien podría mirar por encima de su hombro un día y decir: "¡Guau, ese es un código hermoso!"

Lo simple es mejor que lo complejo.

Si tiene que elegir entre una solución simple y una compleja, y ambas trabajo, use la solución simple. Su código será más fácil de mantener y será más fácil para usted y para otros desarrollar ese código más adelante.

Complejo es mejor que complicado.

La vida real es complicada y, a veces, una solución simple a un problema es inalcanzable. En ese caso, use la solución más simple que funcione.

La legibilidad cuenta.

Incluso cuando su código sea complejo, trate de hacerlo legible. Cuando estas trabajando en un proyecto que involucra una codificación compleja, concéntrese en escribir comentarios informativos para ese código.

Debe haber una, y preferiblemente solo una, forma obvia de hacerlo.

Si se le pide a dos programadores de Python que resuelvan el mismo problema, deberían encontrar soluciones bastante compatibles. Esto no quiere decir que no haya espacio para la creatividad en la programación. ¡De lo contrario! Pero gran parte de la programación consiste en utilizar enfoques pequeños y comunes para situaciones simples dentro de un proyecto más grande y creativo. Los aspectos prácticos de sus programas deberían tener sentido para otros programadores de Python.

Ahora es mejor que nunca.

Podrías pasar el resto de tu vida aprendiendo todas las complejidades de Python y de la programación en general, pero nunca completarías ningún proyecto. No intente escribir un código perfecto; escriba código que funcione y luego decida si mejorar su código para ese proyecto o pasar a algo nuevo.

A medida que continúe con el próximo capítulo y comience a profundizar en temas más complicados, intente tener en cuenta esta filosofía de simplicidad y claridad. Los programadores experimentados respetarán más su código y estarán felices de brindarle comentarios y colaborar con usted en proyectos interesantes.

Inténtalo tú mismo

2-11. Zen of Python: Ingrese **importar esto** en una sesión de terminal de Python y hojee los principios adicionales.

Resumen

En este capítulo aprendiste a trabajar con variables. Aprendió a usar nombres de variables descriptivos y cómo resolver errores de nombre y errores de sintaxis cuando surgen. Aprendió qué son las cadenas y cómo mostrarlas en minúsculas, mayúsculas y título. Comenzó a usar espacios en blanco para organizar la salida ordenadamente y aprendió a eliminar los espacios en blanco innecesarios de diferentes partes de una cadena. Comenzó a trabajar con números enteros y flotantes, y aprendió algunas de las formas en que puede trabajar con datos numéricos. También aprendió a escribir comentarios explicativos para que su código sea más fácil de leer para usted y para otros. Finalmente, lee sobre la filosofía de mantener su código lo más simple posible, siempre que sea posible.

En el Capítulo 3, aprenderá a almacenar colecciones de información en estructuras de datos llamadas *listas.* Aprenderá a trabajar a través de una lista, manipulando cualquier información en esa lista.

3

Introducción a las listas

En este capítulo y el siguiente, aprenderá qué son las listas y cómo empezar a trabajar con los elementos de una lista. Las listas le permiten almacenar conjuntos de información en un solo lugar, ya sea que tenga solo unos pocos elementos o millones de elementos. Las listas son una de las características más poderosas de Python, fácilmente accesibles para los nuevos programadores, y unen muchos conceptos importantes en la programación.

¿Qué es una lista?

Una *lista* es una colección de elementos en un orden particular. Puede hacer una lista que incluya las letras del alfabeto, los dígitos del 0 al 9 o los nombres de todas las personas de su familia. Puedes poner lo que quieras en una lista, y

los elementos de su lista no tienen que estar relacionados de ninguna manera en particular. Debido a que una lista generalmente contiene más de un elemento, es una buena idea hacer que el nombre de su lista sea plural, como letras, dígitos o nombres.

En Python, los corchetes ([]) indican una lista y los elementos individuales de la lista están separados por comas. Aquí hay un ejemplo simple de una lista que contiene algunos tipos de bicicletas:

bicicletas.py

```
bicicletas = ['trek', 'cannondale', 'redline', 'specialized']
imprimir (bicicletas)
```

Si le pide a Python que imprima una lista, Python devuelve su representación del lista, incluidos los corchetes:

```
['trek', 'cannondale', 'redline', 'especializado']
```

Debido a que este no es el resultado que desea que vean sus usuarios, aprendamos cómo para acceder a los elementos individuales de una lista.

Acceso a elementos en una lista

Las listas son colecciones ordenadas, por lo que puede acceder a cualquier elemento de una lista diciéndole a Python la posición o el *índice* del elemento deseado. Para acceder a un elemento de una lista, escriba el nombre de la lista seguido del índice del elemento entre corchetes.

Por ejemplo, saquemos la primera bicicleta de la lista bicicletas:

```
bicicletas = ['trek', 'cannondale', 'redline', 'specialized']
u imprime(bicicletas[0])
```

La sintaxis para esto se muestra en u. Cuando pedimos un solo artículo de un list, Python devuelve solo ese elemento sin corchetes:

```
emigrar
```

Este es el resultado que desea que vean sus usuarios: una salida limpia y con un formato ordenado.

También puede usar los métodos de cadena del Capítulo 2 en cualquier elemento de esta lista. Por ejemplo, puede formatear el elemento 'trek' de forma más clara usando el método title() :

```
bicicletas = ['trek', 'cannondale', 'redline', 'specialized']
imprimir (bicicletas [0]. título ())
```

Este ejemplo produce el mismo resultado que el ejemplo anterior, excepto que 'Trek' está en mayúsculas.

Las posiciones de índice comienzan en 0, no en 1

Python considera que el primer elemento de una lista está en la posición 0, no en la posición 1. Esto es cierto para la mayoría de los lenguajes de programación, y la razón tiene que ver con cómo se implementan las operaciones de lista en un nivel inferior. Si recibe resultados inesperados, determine si está haciendo una comparación simple
error.

El segundo elemento de una lista tiene un índice de 1. Usando este sistema de conteo, puede obtener cualquier elemento que desee de una lista restando uno de su posición en la lista. Por ejemplo, para acceder al cuarto elemento de una lista, solicita el elemento en el índice 3.

Lo siguiente pregunta por las bicicletas en el índice 1 y el índice 3:

```
bicicletas = ['trek', 'cannondale', 'redline', 'specialized']
imprimir(bicicletas[1])
imprimir(bicicletas[3])
```

Este código devuelve la segunda y cuarta bicicleta de la lista:

```
cannondale
especializado
```

Python tiene una sintaxis especial para acceder al último elemento de una lista. por preguntar Al buscar el elemento en el índice -1, Python siempre devuelve el último elemento de la lista:

```
bicicletas = ['trek', 'cannondale', 'redline', 'specialized']
imprimir(bicicletas[-1])
```

Este código devuelve el valor 'especializado'. Esta sintaxis es bastante útil, porque a menudo querrá acceder a los últimos elementos de una lista sin saber exactamente cuánto tiempo tiene la lista. Esta convención se extiende también a otros valores de índice negativos. El índice -2 devuelve el segundo elemento desde el final de la lista, el índice -3 devuelve el tercer elemento desde el final y así sucesivamente.

Uso de valores individuales de una lista

Puede utilizar valores individuales de una lista como lo haría con cualquier otra variable. Por ejemplo, puede usar f-strings para crear un mensaje basado en un valor de una lista.

Intentemos sacar la primera bicicleta de la lista y componer un mensaje usando ese valor.

```
bicicletas = ['trek', 'cannondale', 'redline', 'specialized']
u message = f"Mi primera bicicleta fue una {bicicletas[0].title()}."

imprimir (mensaje)
```

En u, construimos una oración usando el valor en bicicletas[0] y lo asignamos a la variable mensaje. El resultado es una oración simple sobre la primera bicicleta en la lista:

```
Mi primera bicicleta fue una Trek.
```

Inténtalo tú mismo

Pruebe estos programas breves para obtener experiencia de primera mano con las listas de Python. Es posible que desee crear una nueva carpeta para los ejercicios de cada capítulo para mantenerlos organizados.

3-1. Nombres: almacene los nombres de algunos de sus amigos en una lista llamada nombres. Imprima el nombre de cada persona accediendo a cada elemento de la lista, uno a la vez.

3-2. Saludos: Comience con la lista que usó en el Ejercicio 3-1, pero en lugar de solo imprimir el nombre de cada persona, imprima un mensaje para ellos. El texto de cada mensaje debe ser el mismo, pero cada mensaje debe estar personalizado con el nombre de la persona.

3-3. Su propia lista: Piense en su medio de transporte favorito, como una motocicleta o un automóvil, y haga una lista que almacene varios ejemplos. Use su lista para imprimir una serie de afirmaciones sobre estos elementos, como "Me gustaría tener una motocicleta Honda".

Cambio, adición y eliminación de elementos

La mayoría de las listas que cree serán dinámicas, lo que significa que creará una lista y luego agregará y eliminará elementos de ella a medida que su programa sigue su curso. Por ejemplo, puede crear un juego en el que un jugador tenga que disparar a los extraterrestres desde el cielo. Puede almacenar el conjunto inicial de alienígenas en una lista y luego eliminar un alienígena de la lista cada vez que uno es derribado. Cada vez que aparece un nuevo extraterrestre en la pantalla, lo agregas a la lista. Tu lista de alienígenas aumentará y disminuirá en longitud a lo largo del juego.

Modificación de elementos en una lista

La sintaxis para modificar un elemento es similar a la sintaxis para acceder a un elemento en una lista. Para cambiar un elemento, use el nombre de la lista seguido del índice del elemento que desea cambiar y luego proporcione el nuevo valor que desea que tenga ese elemento.

Por ejemplo, supongamos que tenemos una lista de motocicletas y el primer elemento en la lista es 'honda'. ¿Cómo cambiaríamos el valor de este primer artículo?

```
motos.py u motos = ['honda', 'yamaha', 'suzuki']
          imprimir (motos)

        v motos[0] = 'ducati'
          imprimir (motos)
```

El código en u define la lista original, con 'honda' como primer elemento. El código en v cambia el valor del primer elemento a 'ducati'. El resultado muestra que el primer elemento se ha cambiado y el resto de la lista permanece igual:

```
['honda','yamaha','suzuki']
['ducati', 'yamaha', 'suzuki']
```

Puede cambiar el valor de cualquier elemento de una lista, no solo del primero.

Adición de elementos a una lista

Es posible que desee agregar un nuevo elemento a una lista por muchas razones. Por ejemplo, es posible que desee hacer que aparezcan nuevos extraterrestres en un juego, agregar nuevos datos a una visualización o agregar nuevos usuarios registrados a un sitio web que ha creado. Python proporciona varias formas de agregar nuevos datos a las listas existentes.

Agregar elementos al final de una lista

La forma más sencilla de agregar un nuevo elemento a una lista es *agregar* el elemento a la lista. Cuando agrega un elemento a una lista, el nuevo elemento se agrega al final de la lista. Usando la misma lista que teníamos en el ejemplo anterior, agregaremos el nuevo elemento 'ducati' al final de la lista:

```
  motos = ['honda', 'yamaha', 'suzuki']
  imprimir (motos)

u motocicletas.append('ducati')
  imprimir (motos)
```

El método append() en u agrega 'ducati' al final de la lista sin afectando a cualquiera de los otros elementos de la lista:

```
['honda','yamaha','suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

El método append() facilita la creación dinámica de listas. Por ejemplo, puede comenzar con una lista vacía y luego agregar elementos a la lista mediante una serie de llamadas append() . Usando una lista vacía, agreguemos los elementos 'honda', 'yamaha' y 'suzuki' a la lista:

```
motos = []

motocicletas.append('honda')
motocicletas.append('yamaha')
motocicletas.append('suzuki')

imprimir (motos)
```

La lista resultante tiene exactamente el mismo aspecto que las listas de los ejemplos anteriores:

```
['honda','yamaha','suzuki']
```

La creación de listas de esta manera es muy común, porque a menudo no sabrá los datos que sus usuarios desean almacenar en un programa hasta después de que el programa se esté ejecutando. Para que sus usuarios tengan el control, comience definiendo una lista vacía que contendrá los valores de los usuarios. Luego agregue cada nuevo valor proporcionado a la lista que acaba de crear.

Insertar elementos en una lista

Puede agregar un nuevo elemento en cualquier posición de su lista usando insert() método. Para ello, especifique el índice del nuevo elemento y el valor del nuevo elemento.

```
motos = ['honda', 'yamaha', 'suzuki']

u motocicletas.insertar(0, 'ducati')
imprimir (motos)
```

En este ejemplo, el código en u inserta el valor 'ducati' al principio de la lista. El método insert() abre un espacio en la posición 0 y almacena el valor 'ducati' en esa ubicación. Esta operación desplaza todos los demás valores de la lista una posición a la derecha:

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

Eliminación de elementos de una lista

A menudo, deseará eliminar un elemento o un conjunto de elementos de una lista. Por ejemplo, cuando un jugador dispara a un alienígena desde el cielo, lo más probable es que quieras eliminarlo de la lista de alienígenas activos. O cuando un usuario

decide cancelar su cuenta en una aplicación web que usted creó, querrá eliminar a ese usuario de la lista de usuarios activos. Puede eliminar un elemento según su posición en la lista o según su valor.

Eliminación de un elemento mediante la instrucción del

Si conoce la posición del elemento que desea eliminar de una lista, puede usar la instrucción del .

```
motos = ['honda', 'yamaha', 'suzuki']
print(motos)

u del motos[0]
print(motos)
```

El código en u usa del para eliminar el primer elemento, 'honda', de la lista de motocicletas:

```
['honda','yamaha','suzuki']
['yamaha', 'suzuki']
```

Puede eliminar un elemento de cualquier posición en una lista utilizando la instrucción del si conoce su índice. Por ejemplo, aquí se explica cómo eliminar el segundo elemento, 'yamaha', de la lista:

```
motos = ['honda', 'yamaha', 'suzuki']
imprimir (motos)

del motos[1]
print(motos)
```

La segunda motocicleta se elimina de la lista:

```
['honda','yamaha','suzuki']
['honda', 'suzuki']
```

En ambos ejemplos, ya no puede acceder al valor que se eliminó de la lista después de usar la instrucción del .

Eliminación de un elemento mediante el método pop()

A veces querrá usar el valor de un elemento después de eliminarlo de una lista. Por ejemplo, es posible que desee obtener la posición *x* e *y* de un alienígena que acaba de ser derribado, para poder dibujar una explosión en esa posición. En una aplicación web, es posible que desee eliminar un usuario de una lista de miembros activos y luego agregar ese usuario a una lista de miembros inactivos.

El método pop() elimina el último elemento de una lista, pero le permite trabajar con ese elemento después de eliminarlo. El término *pop* proviene de pensar en una lista como una pila de elementos y sacar un elemento de la parte superior de la pila. En esta analogía, la parte superior de una pila corresponde al final de una lista.

Saquemos una motocicleta de la lista de motocicletas:

```
u motos = ['honda', 'yamaha', 'suzuki']
  imprimir (motos)

v motocicleta_reventada = motocicletas.pop() w
print(motocicletas)
x imprimir (motocicleta reventada)
```

Comenzamos definiendo e imprimiendo la lista de motocicletas en u. En v hacemos estallar un valor de la lista y almacenar ese valor en la variable popped_motorcycle. Imprimimos la lista en w para mostrar que se ha eliminado un valor de la lista. Luego imprimimos el valor extraído en x para demostrar que todavía tenemos acceso al valor que se eliminó.

El resultado muestra que el valor 'suzuki' se eliminó del final de la lista y ahora está asignado a la variable popped_motorcycle:

```
['honda','yamaha','suzuki']
['honda','yamaha']
suzuki
```

¿Cómo podría ser útil este método pop() ? Imagina que las motos de la lista están almacenadas en orden cronológico según cuando las tuvimos. Si este es el caso, podemos usar el método pop() para imprimir una declaración sobre la última motocicleta que compramos:

```
motos = ['honda', 'yamaha', 'suzuki']

última_propiedad = motocicletas.pop()
print(f"La última motocicleta que tuve fue una {última_propiedad.título()}").
```

El resultado es una oración simple sobre la motocicleta más reciente que tuvimos:

```
La última motocicleta que tuve fue una Suzuki.
```

Reventar elementos desde cualquier posición en una lista

Puede usar pop() para eliminar un elemento de cualquier posición en una lista al incluir el índice del elemento que desea eliminar entre paréntesis.

```
motos = ['honda', 'yamaha', 'suzuki']

u primera_propiedad = motocicletas.pop(0)
v print(f"La primera motocicleta que tuve fue una {primera_propiedad.título()}").
```

Comenzamos por mostrar la primera motocicleta en la lista en u, y luego imprimimos un mensaje sobre esa motocicleta en v. El resultado es una oración simple que describe la primera motocicleta que tuve:

```
La primera motocicleta que tuve fue una Honda.
```

Recuerde que cada vez que usa pop(), el elemento con el que trabaja ya no se almacena en la lista.

Si no está seguro si usar la declaración del o el método pop() , aquí hay una manera simple de decidir: cuando desee eliminar un elemento de una lista y no usar ese elemento de ninguna manera, use la declaración del ; si desea usar un elemento a medida que lo elimina, use el método pop() .

Eliminación de un artículo por valor

A veces no sabrá la posición del valor que desea eliminar de una lista. Si solo conoce el valor del elemento que desea eliminar, puede usar el método remove() .

Por ejemplo, supongamos que queremos eliminar el valor 'ducati' de la lista de motocicletas.

```
motos = ['honda', 'yamaha', 'suzuki', 'ducati'] print(motos)


u motos.remove('ducati')
  imprimir (motos)
```

El código en u le dice a Python que averigüe dónde aparece 'ducati' en la lista y elimine ese elemento:

```
['honda', 'yamaha', 'suzuki', 'ducati'] ['honda',
'yamaha', 'suzuki']
```

También puede usar el método remove() para trabajar con un valor que se está eliminando de una lista. Eliminemos el valor 'ducati' e imprimamos un motivo para eliminarlo de la lista:

```
u motos = ['honda', 'yamaha', 'suzuki', 'ducati']
  imprimir (motos)

v demasiado_caro = 'ducati'
w motocicletas.remove(demasiado_caro)
  imprimir (motos)
x print(f"\nA {demasiado_caro.title()} es demasiado caro para mí.")
```

Después de definir la lista en u, asignamos el valor 'ducati' a una variable llamada demasiado_caro v. Luego usamos esta variable para decirle a Python qué valor eliminar de la lista en w. En x se ha eliminado el valor 'ducati'

de la lista, pero todavía se puede acceder a través de la variable too_caro, lo que nos permite imprimir una declaración sobre por qué eliminamos 'ducati' de la lista de motocicletas:

```
['honda', 'yamaha', 'suzuki', 'ducati'] ['honda', 'yamaha',
'suzuki']

Una Ducati es demasiado cara para mí.
```

Nota método *remove() elimina solo la primera aparición del valor que especifique. Si existe la posibilidad de que el valor aparezca más de una vez en la lista, deberá usar un bucle para asegurarse de que se eliminen todas las apariciones del valor. Aprenderá a hacer esto en el Capítulo 7.*

Inténtalo tú mismo

Los siguientes ejercicios son un poco más complejos que los del Capítulo 2, pero le brindan la oportunidad de usar listas en todas las formas descritas.

3-4. Lista de invitados: Si pudieras invitar a alguien, vivo o fallecido, a cenar, ¿a quién invitarías? Haz una lista que incluya al menos tres personas a las que te gustaría invitar a cenar. Luego use su lista para imprimir un mensaje para cada persona, invitándolos a cenar.

3-5. Cambiar la lista de invitados: acaba de enterarse de que uno de sus invitados no puede asistir a la cena, por lo que debe enviar un nuevo conjunto de invitaciones. Tendrás que pensar en alguien más a quien invitar.

- Comience con su programa del Ejercicio 3-4. Agregue una llamada print() al final de su programa que indique el nombre del invitado que no puede asistir.
- Modifica tu lista, reemplazando el nombre del invitado que no puede asistir con el nombre de la nueva persona que estás invitando.
- Imprima un segundo conjunto de mensajes de invitación, uno para cada persona que todavía está en tu lista

3-6. Más invitados: Acabas de encontrar una mesa de comedor más grande, por lo que ahora hay más espacio disponible. Piense en tres invitados más para invitar a cenar.

- Comience con su programa del Ejercicio 3-4 o del Ejercicio 3-5. Agregar una impresión () llame al final de su programa para informar a las personas que encontró una mesa más grande para la cena.
- Utilice insert() para agregar un nuevo invitado al comienzo de su lista.
- Utilice insert() para agregar un nuevo invitado al medio de su lista.
- Utilice append() para agregar un nuevo invitado al final de su lista.
- Imprima un nuevo conjunto de mensajes de invitación, uno para cada persona en su lista.

3-7. La lista de invitados se reduce: acaba de enterarse de que su nueva mesa no llegará a tiempo para la cena y solo tiene espacio para dos invitados.

- Comience con su programa del Ejercicio 3-6. Agrega una nueva línea que imprima un mensaje diciendo que solo puedes invitar a dos personas a cenar.
- Utilice pop() para eliminar invitados de su lista de uno en uno hasta que solo queden dos. los nombres permanecen en su lista. Cada vez que seleccione un nombre de su lista, imprima un mensaje para esa persona haciéndole saber que lamenta no poder invitarla a cenar.
- Imprima un mensaje para cada una de las dos personas que todavía están en su lista, haciéndoles saber que todavía están invitados.
- Utilice del para eliminar los dos últimos nombres de su lista, de modo que tenga una lista vacía. Imprima su lista para asegurarse de que realmente tiene una lista vacía al final de su programa.

Organizar una lista

A menudo, sus listas se crearán en un orden impredecible, porque no siempre puede controlar el orden en que los usuarios proporcionan sus datos. Aunque esto es inevitable en la mayoría de las circunstancias, con frecuencia querrá presentar su información en un orden particular. Algunas veces querrá conservar el orden original de su lista y otras veces querrá cambiar el orden original. Python proporciona varias formas diferentes de organizar sus listas, según la situación.

Ordenar una lista de forma permanente con el método sort()

El método sort() de Python hace que sea relativamente fácil ordenar una lista. Imagina que tenemos una lista de autos y queremos cambiar el orden de la lista para almacenarlos alfabéticamente. Para simplificar la tarea, supongamos que todos los valores de la lista están en minúsculas.

autos.py

```
coches = ['bmw','audi','toyota','subaru']
u coches.sort()
imprimir (coches)
```

El método sort() , que se muestra en u, cambia el orden de la lista de forma permanente. Los autos ahora están en orden alfabético y nunca podremos volver al orden original:

```
['audi','bmw','subaru','toyota']
```

También puede ordenar esta lista en orden alfabético inverso pasando el argumento reverse=True al método sort() . El siguiente ejemplo ordena la lista de autos en orden alfabético inverso:

```
coches = ['bmw','audi','toyota','subaru']
autos.sort(reverse=True)
imprimir (coches)
```

Nuevamente, el orden de la lista cambia permanentemente:

```
['toyota','subaru','bmw','audi']
```

Ordenar una lista temporalmente con la función sorted()

Para mantener el orden original de una lista pero presentarla ordenada, puede usar la función sorted() . La función sorted() le permite mostrar su lista en un orden particular pero no afecta el orden real de la lista.

Probemos esta función en la lista de autos.

```
coches = ['bmw','audi','toyota','subaru']

u print("Esta es la lista original:")
imprimir (coches)

v print("\nEsta es la lista ordenada:")
imprimir (ordenado (coches))

w print("\nAquí está de nuevo la lista original:")
imprimir (coches)
```

Primero imprimimos la lista en su orden original en u y luego en orden alfabético en v. Después de mostrar la lista en el nuevo orden, mostramos que la lista todavía está almacenada en su orden original en w.

```
Aquí está la lista original:
['bmw', 'audi', 'toyota', 'subaru']

Aquí está la lista
ordenada: ['audi', 'bmw', 'subaru', 'toyota']

x Aquí está la lista original de nuevo:
['bmw','audi','toyota','subaru']
```

Observe que la lista aún existe en su orden original en x después de sorted() se ha utilizado la función. La función sorted() también puede aceptar un reverse=True argumento si desea mostrar una lista en orden alfabético inverso.

Ordenar una lista alfabéticamente es un poco más complicado cuando todos los valores no están en minúsculas. Hay varias formas de interpretar las letras mayúsculas al determinar un orden de clasificación, y especificar el orden exacto puede ser más complejo de lo que queremos tratar en este momento. Sin embargo, la mayoría de los enfoques de clasificación se basarán directamente en lo que aprendió en esta sección.

Imprimir una lista en orden inverso

Para invertir el orden original de una lista, puede utilizar el método reverse() . Si originalmente almacenáramos la lista de autos en orden cronológico según cuando los poseíamos, podríamos reorganizar fácilmente la lista en orden cronológico inverso:

```
coches = ['bmw','audi','toyota','subaru']
imprimir (coches)

coches.reverse()
imprimir (coches)
```

Tenga en cuenta que reverse() no ordena alfabéticamente hacia atrás; simplemente invierte el orden de la lista:

```
['bmw','audi','toyota','subaru']
['subaru','toyota','audi','bmw']
```

El método reverse() cambia el orden de una lista de forma permanente, pero puede volver al orden original en cualquier momento aplicando reverse() a la misma lista por segunda vez.

Encontrar la longitud de una lista

Puede encontrar rápidamente la longitud de una lista usando la función len() . La lista de este ejemplo tiene cuatro elementos, por lo que su longitud es 4:

```
>>> coches = ['bmw','audi','toyota','subaru']
>>> len(coches)
4
```

Encontrará len() útil cuando necesite identificar la cantidad de extraterrestres que aún deben ser derribados en un juego, determinar la cantidad de datos que debe administrar en una visualización o averiguar la cantidad de usuarios registrados en un sitio web, entre otras tareas.

Python cuenta los elementos de una lista que comienzan con uno, por lo que no debería encontrarse con ningún error de uno en uno al determinar la longitud de una lista.

Inténtalo tú mismo

3-8. Ver el mundo: piensa en al menos cinco lugares del mundo que te gustaría visitar.
visitar.

- Almacenar las ubicaciones en una lista. Asegúrese de que la lista no esté en orden alfabético.
- Imprima su lista en su orden original. No se preocupe por imprimir la lista ordenadamente, simplemente imprímala como una lista de Python sin formato.
- Use sorted() para imprimir su lista en orden alfabético sin modificar el lista real.
- Muestre que su lista todavía está en su orden original imprimiéndola.
- Use sorted() para imprimir su lista en orden alfabético inverso sin cambiar el orden de la lista original.
- Muestre que su lista todavía está en su orden original imprimiéndola de nuevo.
- Utilice reverse() para cambiar el orden de su lista. Imprima la lista para mostrar que su orden ha cambiado.
- Use reverse() para cambiar el orden de su lista nuevamente. Imprima la lista para mostrar que ha vuelto a su orden original.
- Use sort() para cambiar su lista para que se almacene en orden alfabético. Imprima la lista para mostrar que su orden ha sido cambiado.
- Use sort() para cambiar su lista para que se almacene en orden alfabético inverso. Imprima la lista para mostrar que su orden ha cambiado.

3-9. Invitados a cenar: Trabajando con uno de los programas de los Ejercicios 3-4 a 3-7 (página 42), use len() para imprimir un mensaje que indique el número de personas que está invitando a cenar.

3-10. Cada función: piensa en algo que podrías almacenar en una lista. Por ejemplo, podría hacer una lista de montañas, ríos, países, ciudades, idiomas o cualquier otra cosa que desee. Escriba un programa que cree una lista que contenga estos elementos y luego use cada función presentada en este capítulo al menos una vez.

Evitar errores de índice al trabajar con listas

Es común ver un tipo de error cuando se trabaja con listas por primera vez. Supongamos que tiene una lista con tres elementos y solicita el cuarto elemento:

motos.py

```
motos = ['honda', 'yamaha', 'suzuki'] print(motos[3])
```

Este ejemplo da como resultado un *error de índice:*

```
Rastreo (llamadas recientes más última):
  Archivo "motocicletas.py", línea 2, en <módulo>
    imprimir(motos[3])
IndexError: índice de lista fuera de rango
```

Python intenta proporcionarle el elemento en el índice 3. Pero cuando busca en la lista, ningún elemento en motocicletas tiene un índice de 3. Debido a la naturaleza de la indexación en las listas, este error es típico. La gente piensa que el tercer elemento es el elemento número 3, porque comienzan a contar en 1. Pero en Python, el tercer elemento es el número 2, porque comienza a indexarse en 0.

Un error de índice significa que Python no puede encontrar un elemento en el índice que solicitó. Si ocurre un error de índice en su programa, intente ajustar el índice que está solicitando en uno. Luego ejecute el programa nuevamente para ver si los resultados son correctos

Tenga en cuenta que siempre que quiera acceder al último elemento de una lista, use el índice -1. Esto siempre funcionará, incluso si su lista ha cambiado de tamaño desde la última vez que accedió a ella:

```
motos = ['honda', 'yamaha', 'suzuki']
imprimir(motos[-1])
```

El índice -1 siempre devuelve el último elemento de una lista, en este caso el valor 'suzuki':

```
'suzuki'
```

La única vez que este enfoque causará un error es cuando solicita el último elemento de una lista vacía:

```
motos = []
print(motos[-1])
```

No hay elementos en las motocicletas, por lo que Python devuelve otro error de índice:

```
Rastreo (última llamada más reciente):
  Archivo "motorcyles.py", línea 3, en <módulo>
    imprimir(motos[-1])
IndexError: índice de lista fuera de rango
```

Si se produce un error de índice y no sabe cómo resolverlo, intente imprimir su lista o simplemente imprima la longitud de su lista. Su lista puede verse muy diferente de lo que pensaba, especialmente si su programa la ha administrado dinámicamente. Ver la lista real, o la cantidad exacta de elementos en su lista, puede ayudarlo a resolver tales errores lógicos.

Inténtalo tú mismo

3-11. Error intencional: si aún no ha recibido un error de índice en uno de sus programas, intente que suceda. Cambie un índice en uno de sus programas para producir un error de índice. Asegúrese de corregir el error antes de cerrar el programa.

Resumen

En este capítulo aprendió qué son las listas y cómo trabajar con los elementos individuales de una lista. Aprendió cómo definir una lista y cómo agregar y eliminar elementos. Aprendió a ordenar listas de forma permanente y temporal con fines de visualización. También aprendió cómo encontrar la longitud de una lista y cómo evitar errores de índice cuando trabaja con listas.

En el Capítulo 4, aprenderá cómo trabajar con elementos en una lista de manera más eficiente. Al recorrer cada elemento de una lista con solo unas pocas líneas de código, podrá trabajar de manera eficiente, incluso cuando su lista contenga miles o millones de elementos.

4

Trabajar con listas

En el Capítulo 3 aprendió cómo hacer una lista simple y aprendió a trabajar con los elementos individuales en una lista. en este capitulo Más adelante aprenderá a *recorrer* una lista completa utilizando solo unas pocas líneas de código, independientemente de la longitud de la lista. El bucle le permite realizar la misma acción, o conjunto de acciones, con cada elemento de una lista. Como resultado, podrá trabajar de manera eficiente con listas de cualquier longitud, incluidas aquellas con miles o incluso millones de elementos.

Bucle a través de una lista completa

A menudo querrá recorrer todas las entradas de una lista, realizando la misma tarea con cada elemento. Por ejemplo, en un juego, es posible que desee mover todos los elementos de la pantalla en la misma cantidad, o en una lista de números, es posible que desee realizar la misma operación estadística en cada elemento. O tal vez desee mostrar cada título de una lista de artículos en un sitio web. Cuando desee realizar la misma acción con todos los elementos de una lista, puede utilizar el bucle for de Python.

Digamos que tenemos una lista de nombres de magos y queremos imprimir cada nombre en la lista. Podríamos hacer esto recuperando cada nombre de la lista individualmente, pero este enfoque podría causar varios problemas. Por un lado, sería repetitivo hacer esto con una larga lista de nombres. Además, tendríamos que cambiar nuestro código cada vez que cambiara la longitud de la lista. Un bucle for evita ambos problemas al permitir que Python los administre internamente.

Usemos un ciclo for para imprimir cada nombre en una lista de magos:

```
magos.py u magos = ['alice', 'david', 'carolina'] v for mago in
          magos: w print(mago)
```

Comenzamos definiendo una lista en u, tal como lo hicimos en el Capítulo 3. En v, definimos un bucle for . Esta línea le dice a Python que extraiga un nombre de la lista de magos y lo asocie con la variable mago. En w le decimos a Python que imprima el nombre que acaba de ser asignado al mago. Python luego repite las líneas v y w, una vez para cada nombre en la lista. Puede ser útil leer este código como "Para cada mago en la lista de magos, escriba el nombre del mago". El resultado es una copia impresa simple de cada nombre en la lista:

```
alicia
david
carolina
```

Una mirada más cercana al bucle

El concepto de bucle es importante porque es una de las formas más comunes en que una computadora automatiza tareas repetitivas. Por ejemplo, en un ciclo simple como el que usamos en *magicians.py,* Python inicialmente lee la primera línea del ciclo:

```
para mago en magos:
```

Esta línea le dice a Python que recupere el primer valor de la lista de magos y asociarlo con la variable mago. Este primer valor es 'alicia'. Python luego lee la siguiente línea:

```
    imprimir (mago)
```

Python imprime el valor actual de magician, que sigue siendo 'alicia'. Porque la lista contiene más valores, Python vuelve a la primera línea del bucle:

```
para mago en magos:
```

Python recupera el siguiente nombre en la lista, 'david', y lo asocia valor con la variable mago. Python luego ejecuta la línea:

```
    imprimir (mago)
```

Python vuelve a imprimir el valor actual de magician , que ahora es 'david'. Python repite todo el bucle una vez más con el último valor de la lista, 'carolina'. Como no hay más valores en la lista, Python pasa a la siguiente línea del programa. En este caso, nada viene después del bucle for , por lo que el programa simplemente finaliza.

Cuando utilice bucles por primera vez, tenga en cuenta que el conjunto de pasos se repite una vez para cada elemento de la lista, sin importar cuántos elementos haya en la lista. Si tiene un millón de elementos en su lista, Python repite estos pasos un millón de veces y, por lo general, muy rápido.

También tenga en cuenta al escribir sus propios bucles for que puede elegir cualquier nombre que desee para la variable temporal que se asociará con cada valor en la lista. Sin embargo, es útil elegir un nombre significativo que represente un solo elemento de la lista. Por ejemplo, esta es una buena forma de iniciar un ciclo for para obtener una lista de gatos, una lista de perros y una lista general de elementos:

```
para gato en gatos:
para perro en perros:
para el artículo en list_of_items:
```

Estas convenciones de nomenclatura pueden ayudarlo a seguir la acción que se realiza en cada elemento dentro de un bucle for . El uso de nombres singulares y plurales puede ayudarlo a identificar si una sección de código funciona con un solo elemento de la lista o con la lista completa.

Hacer más trabajo dentro de un bucle for

Puede hacer casi cualquier cosa con cada elemento en un bucle for . Construyamos sobre el ejemplo anterior imprimiendo un mensaje a cada mago, diciéndoles que realizaron un gran truco:

magos.py

```
magos = ['alicia', 'david', 'carolina'] para mago en
magos: u print(f"{mago.título()}, ¡fue un gran truco!")
```

La única diferencia en este código está en u donde redactamos un mensaje para cada mago, comenzando con el nombre de ese mago. La primera vez que pasa por el bucle, el valor de magician es 'alice', por lo que Python inicia el primer mensaje con el nombre 'Alice'. La segunda vez a través del mensaje comenzará con 'David', y la tercera vez a través del mensaje comenzará con 'Carolina'.

La salida muestra un mensaje personalizado para cada mago en la lista:

```
¡Alice, ese fue un gran truco!
¡David, ese fue un gran truco!
Carolina, ese fue un gran truco!
```

También puede escribir tantas líneas de código como desee en el bucle for . Cada línea con sangría que sigue a la línea para mago en magos se considera *dentro del bucle,* y cada línea con sangría se ejecuta una vez para cada uno .

valor en la lista. Por lo tanto, puede hacer todo el trabajo que desee con cada valor de la lista.

Agreguemos una segunda línea a nuestro mensaje, diciéndole a cada mago que estamos esperando su próximo truco:

```
magos = ['alicia', 'david', 'carolina'] por mago
en magos:
    print(f"{mago.título()}, ¡fue un gran truco!") u print(f"No
puedo esperar a ver tu próximo truco, {mago.título()}.\n")
```

Debido a que hemos aplicado sangría a ambas llamadas a print(), cada línea se ejecutará una vez por cada mago de la lista. La nueva línea ("\n") en la segunda impresión() call u inserta una línea en blanco después de cada pasada por el bucle. Esto crea un conjunto de mensajes que están perfectamente agrupados para cada persona en la lista:

```
¡Alice, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Alice.

¡David, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, David.

Carolina, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.
```

Puede usar tantas líneas como desee en sus bucles for . En la práctica, a menudo le resultará útil realizar varias operaciones diferentes con cada elemento de una lista cuando utilice un bucle for .

Hacer algo después de un bucle for

¿Qué sucede una vez que un ciclo for ha terminado de ejecutarse? Por lo general, querrá resumir un bloque de salida o pasar a otro trabajo que su programa debe realizar.

Cualquier línea de código después del bucle for que no esté sangrada se ejecuta una vez sin repetición. Escribamos un agradecimiento al grupo de magos en su conjunto, agradeciéndoles por ofrecer un excelente espectáculo. Para mostrar este mensaje de grupo después de que se hayan impreso todos los mensajes individuales, colocamos el mensaje de agradecimiento después del bucle for sin sangría:

```
magos = ['alicia', 'david', 'carolina'] por mago
en magos:
    print(f"{mago.title()}, ¡fue un gran truco!")
    print(f"No puedo esperar a ver tu próximo truco, {mago.título()}.\n")

u print("Gracias a todos. ¡Fue un gran espectáculo de magia!")
```

Las dos primeras llamadas a print() se repiten una vez para cada mago en el lista, como viste antes. Sin embargo, debido a que la línea en u no tiene sangría, se imprime solo una vez:

```
¡Alice, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Alice.

¡David, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, David.

Carolina, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.

Gracias a todos. ¡Ese fue un gran espectáculo de magia!
```

Cuando esté procesando datos usando un bucle for , encontrará que esta es una buena manera de resumir una operación que se realizó en un conjunto de datos completo. Por ejemplo, puede usar un bucle for para inicializar un juego ejecutando una lista de caracteres y mostrando cada carácter en la pantalla. Luego, puede escribir código adicional después de este ciclo que muestra un botón *Reproducir ahora* después de que todos los caracteres se hayan dibujado en la pantalla.

Evitar errores de sangría

Python utiliza la sangría para determinar cómo se relaciona una línea o un grupo de líneas con el resto del programa. En los ejemplos anteriores, las líneas que imprimían mensajes a magos individuales formaban parte del ciclo for porque estaban sangradas. El uso de sangría de Python hace que el código sea muy fácil de leer. Básicamente, utiliza espacios en blanco para forzarte a escribir código con un formato limpio y una estructura visual clara. En los programas de Python más largos, notará bloques de código sangrados en algunos niveles diferentes. Estos niveles de sangría lo ayudan a obtener una idea general de la organización general del programa.

A medida que comience a escribir código que se base en una sangría adecuada, deberá estar atento a algunos *errores de sangría comunes.* Por ejemplo, las personas a veces sangran líneas de código que no necesitan sangría u olvidan sangrar líneas que necesitan sangría. Ver ejemplos de estos errores ahora lo ayudará a evitarlos en el futuro y corregirlos cuando aparezcan en sus propios programas.

Examinemos algunos de los errores de sangría más comunes.

Olvidarse de sangrar

Siempre aplique sangría a la línea después de la instrucción for en un bucle. Si lo olvida, Python le recordará:

magos.py

```
magos = ['alicia', 'david', 'carolina'] for mago en
magos: u print(mago)
```

La llamada a print() u debería estar sangrada, pero no lo está. Cuando Python espera un bloque sangrado y no lo encuentra, le permite saber con qué línea tuvo un problema.

```
  Archivo "magos.py", línea 3
    print(mago)
     ^
Error Tabulación: Se esperaba un bloque tabulado
```

Por lo general, puede resolver este tipo de error de sangría sangrando el línea o líneas inmediatamente después de la instrucción for .

Olvidarse de sangrar líneas adicionales

A veces, su bucle se ejecutará sin errores, pero no producirá el resultado esperado. Esto puede suceder cuando intenta realizar varias tareas en un bucle y olvida sangrar algunas de sus líneas.

Por ejemplo, esto es lo que sucede cuando nos olvidamos de sangrar la segunda línea en el ciclo que le dice a cada mago que estamos esperando su próximo truco:

```
magos = ['alicia', 'david', 'carolina']
para mago en magos:
    print(f"{mago.title()}, ¡fue un gran truco!")
u print(f"No puedo esperar a ver tu próximo truco, {mago.título()}.\n")
```

Se supone que la llamada a print() u está sangrada, pero como Python encuentra al menos una línea sangrada después de la instrucción for , no informa de un error. Como resultado, la primera llamada print() se ejecuta una vez para cada nombre de la lista porque está sangrado. La segunda llamada de print() no tiene sangría, por lo que se ejecuta solo una vez después de que el ciclo haya terminado de ejecutarse. Debido a que el valor final asociado con el mago es 'carolina', ella es la única que recibe el mensaje "esperando con ansias el próximo truco":

```
¡Alice, ese fue un gran truco!
¡David, ese fue un gran truco!
Carolina, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.
```

Este es un *error lógico.* La sintaxis es código Python válido, pero el código no produce el resultado deseado porque ocurre un problema en su lógica. Si espera ver una determinada acción repetida una vez para cada elemento de una lista y se ejecuta solo una vez, determine si necesita sangrar simplemente una línea o un grupo de líneas.

sangría innecesariamente

Si sangra accidentalmente una línea que no necesita sangría, Python le informa sobre la sangría inesperada:

```
hello_world.py mensaje = "¡Hola, mundo de Python!"
u imprimes (mensaje)
```

No necesitamos sangrar la llamada print() u, porque no es parte de un ciclo; por lo tanto, Python informa ese error:

```
  Archivo "hello_world.py", línea 2
    imprimir (mensaje)
    ^
IndentationError: sangría inesperada
```

Puede evitar errores de sangría inesperados sangrando solo cuando tenga una razón específica para hacerlo. En los programas que está escribiendo en este momento, las únicas líneas que debe sangrar son las acciones que desea repetir para cada elemento en un bucle for .

Sangría innecesariamente después del bucle

Si sangra accidentalmente el código que debería ejecutarse después de que finalice un bucle, ese código se repetirá una vez para cada elemento de la lista. A veces, esto hace que Python informe de un error, pero a menudo resultará en un error lógico.

Por ejemplo, veamos qué sucede cuando accidentalmente sangramos la línea que agradeció a los magos como grupo por ofrecer un buen espectáculo:

```
magos.py    magos = ['alicia', 'david', 'carolina'] por mago en
            magos:
                print(f"{mago.title()}, ¡fue un gran truco!")
                print(f"No puedo esperar a ver tu próximo truco, {mago.título()}.\n")

u print("¡Gracias a todos, fue un gran espectáculo de magia!")
```

Debido a que la línea en u está sangrada, se imprime una vez para cada persona en la lista, como se muestra aquí:

```
¡Alice, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Alice.

¡Gracias a todos, fue un gran espectáculo de magia!
¡David, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, David.

¡Gracias a todos, fue un gran espectáculo de magia!
Carolina, ese fue un gran truco!
No puedo esperar a ver tu próximo truco, Carolina.

¡Gracias a todos, fue un gran espectáculo de magia!
```

Este es otro error lógico, similar al de “Olvidarse de sangrar Líneas adicionales” en la página 54. Debido a que Python no sabe lo que está tratando de lograr con su código, ejecutará todo el código que esté escrito en una sintaxis válida. Si una acción se repite muchas veces cuando debería ejecutarse solo una vez, probablemente necesite eliminar la sangría del código para esa acción.

Olvidando el Colón

Los dos puntos al final de una instrucción for le dicen a Python que interprete la siguiente línea como el comienzo de un ciclo.

```
magos = ['alicia', 'david', 'carolina'] u de mago en
magos
    imprimir (mago)
```

Si olvida accidentalmente los dos puntos, como se muestra en u, obtendrá un error de sintaxis porque Python no sabe lo que está tratando de hacer. Aunque este es un error fácil de solucionar, no siempre es un error fácil de encontrar. Te sorprendería la cantidad de tiempo que dedican los programadores a buscar errores de un solo carácter como este. Dichos errores son difíciles de encontrar porque a menudo solo vemos lo que esperamos ver.

Inténtalo tú mismo

4-1. Pizzas: Piensa en al menos tres tipos de tu pizza favorita. Almacene estos nombres de pizza en una lista y luego use un ciclo for para imprimir el nombre de cada pizza.

- Modifica tu ciclo for para imprimir una oración usando el nombre de la pizza en lugar de imprimir solo el nombre de la pizza. Para cada pizza, debe tener una línea de salida que contenga una declaración simple como Me gusta la pizza de pepperoni.
- Agregue una línea al final de su programa, fuera del bucle for , que indique cuánto te gusta la pizza. El resultado debe constar de tres o más líneas sobre los tipos de pizza que te gustan y luego una oración adicional, como ¡Me encanta la pizza!

4-2. Animales: Piensa en al menos tres animales diferentes que tengan una característica común. Guarde los nombres de estos animales en una lista y luego use un ciclo for para imprimir el nombre de cada animal.

- Modifique su programa para imprimir una declaración sobre cada animal, como Un perro sería una gran mascota.
- Agregue una línea al final de su programa indicando lo que estos animales tienen en común. Podría escribir una oración como ¡Cualquiera de estos animales sería una gran mascota!

Hacer listas numéricas

Existen muchas razones para almacenar un conjunto de números. Por ejemplo, deberá realizar un seguimiento de las posiciones de cada personaje en un juego, y es posible que también desee realizar un seguimiento de las puntuaciones más altas de un jugador. En las visualizaciones de datos, casi siempre trabajará con conjuntos de números, como temperaturas, distancias, tamaños de población o valores de latitud y longitud, entre otros tipos de conjuntos numéricos.

Las listas son ideales para almacenar conjuntos de números, y Python proporciona una variedad de herramientas para ayudarlo a trabajar de manera eficiente con listas de números. Una vez que comprenda cómo usar estas herramientas de manera efectiva, su código funcionará bien incluso cuando sus listas contengan millones de elementos.

Uso de la función range()

La función range() de Python facilita la generación de una serie de números. Por ejemplo, puede usar la función range() para imprimir una serie de números como este:

primero
_numeros.py

```
para valor en rango (1, 5):
    imprimir (valor)
```

Aunque parece que este código debería imprimir los números del 1 al 5, no imprime el número 5:

```
1
2
3
4
```

En este ejemplo, range() imprime solo los números del 1 al 4. Este es otro resultado del comportamiento de desactivación por uno que verá a menudo en los lenguajes de programación. La función range() hace que Python comience a contar en el primer valor que le proporcione y se detenga cuando alcance el segundo valor que proporcione. Debido a que se detiene en ese segundo valor, la salida nunca contiene el valor final, que habría sido 5 en este caso.

Para imprimir los números del 1 al 5, usaría el rango (1, 6):

```
para el valor en el rango (1, 6):
    imprimir (valor)
```

Esta vez, la salida comienza en 1 y termina en 5:

```
1
2
3
4
5
```

Si su resultado es diferente de lo que espera cuando usa range(), intente ajustar su valor final en 1.

También puede pasar range() solo un argumento, y comenzará la secuencia de números en 0. Por ejemplo, range(6) devolvería los números del 0 al 5.

Usando range() para hacer una lista de números

Si desea hacer una lista de números, puede convertir los resultados de range() directamente en una lista usando la función list() . Cuando envuelve list() alrededor de una llamada a la función range() , la salida será una lista de números.

En el ejemplo de la sección anterior, simplemente imprimimos una serie de números. Podemos usar list() para convertir ese mismo conjunto de números en una lista:

```
numeros = lista(rango(1, 6))
imprimir (números)
```

Y este es el resultado:

```
[1, 2, 3, 4, 5]
```

También podemos usar la función range() para decirle a Python que omita números en un rango determinado. Si pasa un tercer argumento a range(), Python usa ese valor como un tamaño de paso al generar números.

Por ejemplo, así es como enumerar los números pares entre 1 y 10:

```
numeros_pares.py numeros_pares = lista(rango(2, 11, 2))
print(numeros_pares)
```

En este ejemplo, la función range() comienza con el valor 2 y luego suma 2 a ese valor. Suma 2 repetidamente hasta que alcanza o pasa el valor final, 11, y produce este resultado:

```
[2, 4, 6, 8, 10]
```

Puedes crear casi cualquier conjunto de números que quieras usando el rango () función. Por ejemplo, considere cómo podría hacer una lista de los primeros 10 números cuadrados (es decir, el cuadrado de cada número entero del 1 al 10). En Python, dos asteriscos (**) representan exponentes. Así es como podrías poner los primeros 10 números cuadrados en una lista:

```
cuadrados.py u cuadrados = []
v para valor en rango (1, 11):
w cuadrado = valor ** 2
x cuadrados.append(cuadrado)

y imprimir (cuadrados)
```

Empezamos con una lista vacía llamada cuadrados u. En v, le decimos a Python que recorra cada valor del 1 al 10 usando la función range() . Dentro del bucle,

el valor actual se eleva a la segunda potencia y se asigna al cuadrado de la variable w. En x, cada nuevo valor de cuadrado se agrega a la lista de cuadrados. Finalmente, cuando el ciclo ha terminado de ejecutarse, se imprime la lista de cuadrados y:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Para escribir este código de manera más concisa, omita la variable temporal cuadrado y agregue cada nuevo valor directamente a la lista:

```
cuadrados = []
para el valor en el rango (1,11):
u cuadrados.append(valor**2)

imprimir (cuadrados)
```

El código en u hace el mismo trabajo que las líneas en w y x en *squares.py.* Cada valor en el bucle se eleva a la segunda potencia y luego se agrega inmediatamente a la lista de cuadrados.

Puede usar cualquiera de estos dos enfoques cuando esté haciendo listas más complejas. A veces, usar una variable temporal hace que su código sea más fácil de leer; otras veces hace que el código sea innecesariamente largo. Concéntrese primero en escribir código que entienda claramente, que haga lo que usted quiere que haga. Luego busque enfoques más eficientes mientras revisa su código.

Estadísticas simples con una lista de números

Algunas funciones de Python son útiles cuando se trabaja con listas de números. Por ejemplo, puede encontrar fácilmente el mínimo, el máximo y la suma de una lista de números:

```
>>> dígitos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(digitos)
0
>>> max(digitos)
9
>>> suma(digitos)
45
```

Los ejemplos en esta sección usan listas cortas de números para encajar fácilmente en el página. Funcionarían igual de bien si su lista contuviera un millón o más de números.

Lista de comprensiones

El enfoque descrito anteriormente para generar los cuadrados de la lista consistía en utilizar tres o cuatro líneas de código. Una *lista de comprensión* le permite generar esta misma lista en una sola línea de código. Una lista por comprensión combina el bucle for y la creación de nuevos elementos en una sola línea, y agrega automáticamente cada elemento nuevo. Las listas de comprensión no siempre se presentan a los principiantes, pero las he incluido aquí porque lo más probable es que las vea tan pronto como empiece a mirar el código de otras personas.

El siguiente ejemplo construye la misma lista de números cuadrados que viste antes pero usa una lista de comprensión:

cuadrados.py

```
cuadrados = [valor**2 para valor en rango (1, 11)]
imprimir (cuadrados)
```

Para usar esta sintaxis, comience con un nombre descriptivo para la lista, como cuadrados. A continuación, abra un conjunto de corchetes y defina la expresión de los valores que desea almacenar en la nueva lista. En este ejemplo, la expresión es valor**2, lo que eleva el valor a la segunda potencia. Luego, escriba un ciclo for para generar los números que desea introducir en la expresión y cierre los corchetes. El bucle for en este ejemplo es for value in range(1, 11), que introduce los valores del 1 al 10 en la expresión value**2. Tenga en cuenta que no se utilizan dos puntos al final de la instrucción for .

El resultado es la misma lista de números cuadrados que viste antes:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Se necesita práctica para escribir sus propias listas de comprensión, pero encontrará vale la pena una vez que te sientas cómodo creando listas ordinarias. Cuando esté escribiendo tres o cuatro líneas de código para generar listas y comience a sentirse repetitivo, considere escribir sus propias comprensiones de listas.

Inténtalo tú mismo

4-3. Contar hasta veinte: use un bucle for para imprimir los números del 1 al 20, inclusive.

4-4. Un millón: haga una lista de los números del uno al millón y luego use un ciclo for para imprimir los números. (Si la salida tarda demasiado, deténgala presionando ctrl-C o cerrando la ventana de salida).

4-5. Suma de un millón: haga una lista de los números del uno al millón y luego use min() y max() para asegurarse de que su lista realmente comience en uno y termine en un millón. Además, use la función sum() para ver qué tan rápido Python puede agregar un millón de números.

4-6. Números impares: use el tercer argumento de la función range() para hacer una lista de los números impares del 1 al 20. Use un bucle for para imprimir cada número.

4-7. Threes: Haga una lista de los múltiplos de 3 de 3 a 30. Use un ciclo for para imprimir los números en su lista.

4-8. Cubos: Un número elevado a la tercera potencia se llama cubo. Por ejemplo, el cubo de 2 se escribe como 2**3 en Python. Haga una lista de los primeros 10 cubos (es decir, el cubo de cada número entero del 1 al 10) y use un ciclo for para imprimir el valor de cada cubo.

4-9. Comprensión de cubos: use una lista de comprensión para generar una lista de los primeros 10 cubos.

Trabajar con parte de una lista

En el Capítulo 3 aprendió cómo acceder a elementos individuales en una lista, y en este capítulo ha estado aprendiendo cómo trabajar con todos los elementos en una lista. También puede trabajar con un grupo específico de elementos en una lista, que Python llama *segmento.*

Cortar una lista

Para hacer un corte, especifica el índice del primer y último elemento con el que desea trabajar. Al igual que con la función range() , Python detiene un elemento antes del segundo índice que especifique. Para generar los tres primeros elementos de una lista, solicitaría los índices 0 a 3, que devolverían los elementos 0, 1 y 2.

El siguiente ejemplo involucra una lista de jugadores en un equipo:

jugadores.py

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
u print(jugadores[0:3])
```

El código en u imprime una parte de esta lista, que incluye solo los tres primeros jugadores. La salida conserva la estructura de la lista e incluye los tres primeros jugadores de la lista:

```
['charles', 'martina', 'miguel']
```

Puede generar cualquier subconjunto de una lista. Por ejemplo, si quiere los elementos segundo, tercero y cuarto de una lista, comenzaría el segmento en el índice 1 y terminaría en el índice 4:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir (jugadores [1: 4])
```

Esta vez el segmento comienza con 'martina' y termina con 'florence':

```
['martina', 'miguel', 'florencia']
```

Si omite el primer índice en un segmento, Python inicia automáticamente su rebanada al principio de la lista:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir (jugadores [: 4])
```

Sin un índice inicial, Python comienza al principio de la lista:

```
['charles', 'martina', 'miguel', 'florencia']
```

Una sintaxis similar funciona si desea un segmento que incluya el final de una lista. Por ejemplo, si desea todos los elementos desde el tercer elemento hasta el último elemento, puede comenzar con el índice 2 y omitir el segundo índice:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir (jugadores [2:])
```

Python devuelve todos los elementos desde el tercer elemento hasta el final de la lista:

```
['miguel', 'florencia', 'eli']
```

Esta sintaxis le permite generar todos los elementos desde cualquier punto de su lista hasta el final, independientemente de la longitud de la lista. Recuerde que un índice negativo devuelve un elemento a cierta distancia del final de una lista; por lo tanto, puede generar cualquier segmento desde el final de una lista. Por ejemplo, si queremos dar salida a los últimos tres jugadores de la lista, podemos usar el segmento players[-3:]:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
imprimir (jugadores [-3:])
```

Esto imprime los nombres de los últimos tres jugadores y continuaría funcionando a medida que la lista de jugadores cambia de tamaño.

Puede incluir un tercer valor entre paréntesis que indique un sector. Si se incluye un tercer valor, esto le dice a Python cuántos elementos omitir entre elementos en el rango especificado.

Bucle a través de una rebanada

Puede usar un segmento en un bucle for si desea recorrer un subconjunto de los elementos de una lista. En el siguiente ejemplo, recorremos los primeros tres jugadores e imprimimos sus nombres como parte de una lista simple:

```
jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']

print("Estos son los tres primeros jugadores de mi equipo:")
u para jugador en jugadores[:3]:
    imprimir (jugador.título())
```

En lugar de recorrer la lista completa de jugadores en u, Python realiza un bucle solo a través de los tres primeros nombres:

```
Aquí están los tres primeros jugadores de mi equipo:
Charles
martina
Miguel
```

Las rebanadas son muy útiles en varias situaciones. Por ejemplo, cuando estás Al crear un juego, puede agregar la puntuación final de un jugador a una lista cada vez que

el jugador termina de jugar. A continuación, puede obtener las tres puntuaciones más altas de un jugador ordenando la lista en orden decreciente y tomando una porción que incluya solo las tres primeras puntuaciones. Cuando trabaja con datos, puede usar segmentos para procesar sus datos en fragmentos de un tamaño específico. O bien, cuando está creando una aplicación web, puede usar segmentos para mostrar información en una serie de páginas con una cantidad adecuada de información en cada página.

Copiar una lista

A menudo, deseará comenzar con una lista existente y crear una lista completamente nueva basada en la primera. Exploremos cómo funciona copiar una lista y examinemos una situación en la que copiar una lista es útil.

Para copiar una lista, puede crear un segmento que incluya la lista original completa omitiendo el primer índice y el segundo índice ([:]). Esto le dice a Python que haga una porción que comience en el primer elemento y termine en el último, produciendo una copia de la lista completa.

Por ejemplo, imagina que tenemos una lista de nuestras comidas favoritas y queremos hacer una lista separada de las comidas que le gustan a un amigo. A este amigo le gusta todo lo que hay en nuestra lista hasta ahora, así que podemos crear su lista copiando la nuestra:

```
comidas.py u my_foods = ['pizza', 'falafel', 'pastel de zanahoria']
v friend_foods = my_foods[:]

print("Mis comidas favoritas son:")
imprimir (mis_alimentos)

print("\nLas comidas favoritas de mi amigo son:")
imprimir (friend_foods)
```

En u hacemos una lista de los alimentos que nos gustan llamada my_foods. En v hacemos una nueva lista llamada friend_foods. Hacemos una copia de my_foods solicitando una porción de my_foods sin especificar ningún índice y almacenamos la copia en friend_foods. Cuando imprimimos cada lista, vemos que ambas contienen los mismos alimentos:

```
Mis comidas favoritas
son: ['pizza', 'falafel', 'carrot cake']

Las comidas favoritas de mi amigo
son: ['pizza', 'falafel', 'carrot cake']
```

Para demostrar que en realidad tenemos dos listas separadas, agregaremos un nuevo alimento a cada lista y mostraremos que cada lista realiza un seguimiento de los alimentos favoritos de la persona adecuada:

```
my_foods = ['pizza', 'falafel', 'carrot cake'] u friend_foods
= my_foods[:]

v my_foods.append('cannoli')
w friend_foods.append('helado')
```

```
print("Mis comidas favoritas son:")
imprimir (mis_alimentos)

print("\nLas comidas favoritas de mi amigo son:")
imprimir (friend_foods)
```

En u copiamos los elementos originales en my_foods a la nueva lista friend_foods, como hicimos en el ejemplo anterior. A continuación, agregamos un nuevo alimento a cada lista: en v agregamos 'cannoli' a my_foods, y en w agregamos 'helado' a friend_foods. Luego imprimimos las dos listas para ver si cada uno de estos alimentos está en la lista adecuada.

```
Mis comidas favoritas son:
x ['pizza', 'falafel', 'carrot cake', 'cannoli']

Las comidas favoritas de mi amiga
son: y ['pizza', 'falafel', 'carrot cake', 'helado']
```

El resultado en x muestra que 'cannoli' ahora aparece en nuestra lista de comidas favoritas pero 'helado' no. En y podemos ver que 'helado' ahora aparece en la lista de nuestros amigos pero 'cannoli' no. Si simplemente hubiéramos establecido friend_foods igual a my_foods, no produciríamos dos listas separadas. Por ejemplo, esto es lo que sucede cuando intenta copiar una lista sin usar una división:

```
my_foods = ['pizza', 'falafel', 'pastel de zanahoria']

# Esto no funciona:
u friend_foods = mis_alimentos

my_foods.append('cannoli')
friend_foods.append('helado')

print("Mis comidas favoritas son:")
imprimir (mis_alimentos)

print("\nLas comidas favoritas de mi amigo son:")
imprimir (friend_foods)
```

En lugar de almacenar una copia de my_foods en friend_foods en ti , configuramos friend _alimentos igual a mis_alimentos. Esta sintaxis en realidad le dice a Python que asocie la nueva variable friend_foods con la lista que ya está asociada con my_foods, por lo que ahora ambas variables apuntan a la misma lista. Como resultado, cuando agregamos 'cannoli' a my_foods, también aparecerá en friend_foods. Del mismo modo , 'helado' aparecerá en ambas listas, aunque parece que se agrega solo a friend_foods.

El resultado muestra que ambas listas son iguales ahora, que no es lo que queríamos:

```
Mis comidas favoritas son:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'helado']
```

```
Las comidas favoritas de mi amigo son: ['pizza',
 'falafel', 'carrot cake', 'cannoli', 'helado']
```

N ota *se preocupe por los detalles de este ejemplo por ahora. Básicamente, si estás tratando de trabaja con una copia de una lista y observa un comportamiento inesperado, asegúrese de copiar la lista mediante un segmento, como hicimos en el primer ejemplo.*

Inténtalo tú mismo

4-10. Rebanadas: usando uno de los programas que escribió en este capítulo, agregue varias líneas al final del programa que hagan lo siguiente:

- Imprimir el mensaje Los tres primeros elementos de la lista son:. Luego use una rebanada para imprima los tres primeros elementos de la lista de ese programa.
- Imprima el mensaje Los tres elementos de la mitad de la lista son:. Usa una rebanada para imprima tres elementos del medio de la lista.
- Imprimir el mensaje Los últimos tres elementos de la lista son:. Utilice una rebanada para imprimir el los tres últimos elementos de la lista.

4-11. Mis pizzas, tus pizzas: comienza con tu programa del ejercicio 4-1 (página 56). Haga una copia de la lista de pizzas y llámela friend_pizzas.
Luego, haz lo siguiente:

- Añadir una nueva pizza a la lista original.
- Añadir una pizza diferente a la lista friend_pizzas.
- Demuestra que tienes dos listas separadas. Imprime el mensaje Mi favorito pizzas are:, y luego use un bucle for para imprimir la primera lista. Imprima el mensaje Las pizzas favoritas de mi amigo son: y luego use un bucle for para imprimir la segunda lista. Asegúrese de que cada pizza nueva se almacene en la lista adecuada.

4-12. Más bucles: todas las versiones de foods.py en esta sección han evitado el uso de bucles al imprimir para ahorrar espacio. Elija una versión de foods.py y escriba dos bucles for para imprimir cada lista de alimentos.

tuplas

Las listas funcionan bien para almacenar colecciones de elementos que pueden cambiar a lo largo de la vida de un programa. La capacidad de modificar listas es particularmente importante cuando trabaja con una lista de usuarios en un sitio web o una lista de personajes en un juego. Sin embargo, a veces querrá crear una lista de elementos que no se pueden cambiar. Las tuplas te permiten hacer precisamente eso. Python se refiere a valores que no pueden cambiar como *inmutables,* y una lista inmutable se llama *tupla.*

Definición de una tupla

Una tupla se parece a una lista, excepto que usa paréntesis en lugar de corchetes. Una vez que define una tupla, puede acceder a elementos individuales utilizando el índice de cada elemento, tal como lo haría con una lista.

Por ejemplo, si tenemos un rectángulo que siempre debe tener un tamaño determinado, podemos asegurarnos de que su tamaño no cambie poniendo las dimensiones en una tupla:

```
dimensiones.py u dimensiones = (200, 50)
               v imprimir (dimensiones [0])
                 imprimir (dimensiones [1])
```

Definimos las dimensiones de la tupla en u, usando paréntesis en lugar de cuadrados soportes. En v imprimimos cada elemento en la tupla individualmente, usando la misma sintaxis que hemos estado usando para acceder a los elementos en una lista:

```
200
50
```

Veamos qué sucede si intentamos cambiar uno de los elementos en las dimensiones de la tupla:

```
  dimensiones = (200, 50)
u dimensiones[0] = 250
```

El código en u intenta cambiar el valor de la primera dimensión, pero Python devuelve un error de tipo. Básicamente, debido a que estamos tratando de alterar una tupla, lo que no se puede hacer con ese tipo de objeto, Python nos dice que no podemos asignar un nuevo valor a un elemento en una tupla:

```
Rastreo (llamadas recientes más última):
  Archivo "dimensions.py", línea 2, en <módulo>
    dimensiones[0] = 250
TypeError: el objeto 'tupla' no admite la asignación de elementos
```

Esto es beneficioso porque queremos que Python genere un error cuando una línea de código intente cambiar las dimensiones del rectángulo.

Las tuplas se definen técnicamente por la presencia de una coma; los paréntesis los hacen parecer más ordenados y legibles. Si desea definir una tupla con un elemento, debe incluir una coma final:

```
mi_t = (3,)
```

A menudo no tiene sentido construir una tupla con un elemento, pero esto puede suceder cuando las tuplas se generan automáticamente.

Recorriendo todos los valores en una tupla

Puede recorrer todos los valores en una tupla usando un bucle for , tal como lo hizo con una lista:

```
dimensiones = (200, 50)
para dimensión en dimensiones:
    imprimir (dimensión)
```

Python devuelve todos los elementos de la tupla, tal como lo haría con una lista:

```
200
50
```

Escribiendo sobre una Tupla

Aunque no puede modificar una tupla, puede asignar un nuevo valor a una variable que representa una tupla. Entonces, si quisiéramos cambiar nuestras dimensiones, podríamos redefinir la tupla completa:

```
u dimensiones = (200, 50)
  imprimir("Dimensiones originales:")
  para dimensión en dimensiones:
      imprimir (dimensión)

v dimensiones = (400, 100)
w print("\nDimensiones modificadas:")
  para dimensión en dimensiones:
      imprimir (dimensión)
```

Las líneas que comienzan en u definen la tupla original e imprimen las dimensiones iniciales. En v, asociamos una nueva tupla con las dimensiones variables. Luego imprimimos las nuevas dimensiones en w. Python no genera ningún error esta vez, porque la reasignación de una variable es válida:

```
Dimensiones originales: 200

50

Dimensiones modificadas:
400
100
```

En comparación con las listas, las tuplas son estructuras de datos simples. Úselos cuando desee almacenar un conjunto de valores que no deben cambiarse a lo largo de la vida de un programa.

Inténtalo tú mismo

4-13. Buffet: Un restaurante estilo buffet ofrece solo cinco alimentos básicos. Piense en cinco alimentos simples y guárdelos en una tupla.

- Utilice un bucle for para imprimir cada comida que ofrece el restaurante.
- Intente modificar uno de los elementos y asegúrese de que Python rechace el cambio.
- El restaurante cambia su menú, reemplazando dos de los artículos con alimentos diferentes. Agregue una línea que reescriba la tupla y luego use un ciclo for para imprimir cada uno de los elementos en el menú revisado.

Estilo de su código

Ahora que está escribiendo programas más largos, vale la pena conocer ideas sobre cómo diseñar su código. Tómese el tiempo para hacer que su código sea lo más fácil de leer posible. Escribir código fácil de leer lo ayuda a realizar un seguimiento de lo que están haciendo sus programas y también ayuda a otros a comprender su código.

Los programadores de Python acordaron una serie de convenciones de estilo para garantizar que el código de todos esté estructurado más o menos de la misma manera. Una vez que haya aprendido a escribir código Python limpio, debería poder comprender la estructura general del código Python de cualquier otra persona, siempre que siga las mismas pautas. Si espera convertirse en un programador profesional en algún momento, debe comenzar a seguir estas pautas lo antes posible para desarrollar buenos hábitos.

La guía de estilo

Cuando alguien quiere hacer un cambio en el lenguaje Python, escribe una *propuesta de mejora de Python (PEP).* Uno de los PEP más antiguos es *PEP* 8, que instruye a los programadores de Python sobre cómo diseñar su código. PEP 8 es bastante largo, pero gran parte se relaciona con estructuras de codificación más complejas que las que has visto hasta ahora.

La guía de estilo de Python se escribió con el entendimiento de que el código se lee con más frecuencia de lo que se escribe. Escribirá su código una vez y luego comenzará a leerlo cuando comience la depuración. Cuando agrega características a un programa, pasará más tiempo leyendo su código. Cuando comparte su código con otros programadores, también lo leerán.

Dada la elección entre escribir código que sea más fácil de escribir o código que sea más fácil de leer, los programadores de Python casi siempre lo alentarán a escribir código que sea más fácil de leer. Las siguientes pautas lo ayudarán a escribir un código claro desde el principio.

Sangría

PEP 8 recomienda utilizar cuatro espacios por nivel de sangría. El uso de cuatro espacios mejora la legibilidad y deja espacio para varios niveles de sangría en cada línea.

En un documento de procesamiento de texto, las personas a menudo usan tabulaciones en lugar de espacios para sangrar. Esto funciona bien para documentos de procesamiento de texto, pero el intérprete de Python se confunde cuando las pestañas se mezclan con espacios. Cada editor de texto proporciona una configuración que le permite usar la tecla de tabulación pero luego convierte cada tabulación a un número determinado de espacios. Definitivamente debe usar su tecla de tabulación , pero también asegúrese de que su editor esté configurado para insertar espacios en lugar de tabulaciones en su documento.

Mezclar tabulaciones y espacios en su archivo puede causar problemas que son muy difíciles de diagnosticar. Si cree que tiene una combinación de tabulaciones y espacios, puede convertir todas las tabulaciones de un archivo en espacios en la mayoría de los editores.

Longitud de la línea

Muchos programadores de Python recomiendan que cada línea tenga menos de 80 caracteres. Históricamente, esta directriz se desarrolló porque la mayoría de las computadoras podían incluir solo 79 caracteres en una sola línea en una ventana de terminal.

Actualmente, las personas pueden colocar líneas mucho más largas en sus pantallas, pero existen otras razones para adherirse a la longitud de línea estándar de 79 caracteres. Los programadores profesionales a menudo tienen varios archivos abiertos en la misma pantalla, y usar la longitud de línea estándar les permite ver líneas completas en dos o tres archivos que están abiertos uno al lado del otro en la pantalla. PEP 8 también recomienda que limite todos sus comentarios a 72 caracteres por línea, porque algunas de las herramientas que generan documentación automática para proyectos más grandes agregan caracteres de formato al comienzo de cada línea comentada.

Las pautas de PEP 8 para la longitud de la línea no son inamovibles y algunos equipos prefieren un límite de 99 caracteres. No se preocupe demasiado por la longitud de línea en su código mientras aprende, pero tenga en cuenta que las personas que trabajan en colaboración casi siempre siguen las pautas de PEP 8. La mayoría de los editores le permiten configurar una señal visual, generalmente una línea vertical en su pantalla, que le muestra dónde están estos límites.

El Apéndice B le muestra cómo configurar su editor de texto para que siempre inserte cuatro espacios cada vez que presione la tecla de *tabulación y muestre una guía vertical para ayudarlo a seguir el límite de 79 caracteres.*

Líneas en blanco

Para agrupar partes de su programa visualmente, use líneas en blanco. Debe usar líneas en blanco para organizar sus archivos, pero no lo haga en exceso. Siguiendo los ejemplos proporcionados en este libro, debe lograr el equilibrio adecuado. Por ejemplo, si tiene cinco líneas de código que crean una lista y luego otras tres líneas que hacen algo con esa lista, es apropiado colocar una línea en blanco entre las dos secciones. Sin embargo, no debe colocar tres o cuatro líneas en blanco entre las dos secciones.

Las líneas en blanco no afectarán la forma en que se ejecuta su código, pero afectarán la legibilidad de su código. El intérprete de Python usa sangría horizontal para interpretar el significado de su código, pero ignora el espacio vertical.

Otras pautas de estilo

PEP 8 tiene muchas recomendaciones de estilo adicionales, pero la mayoría de las líneas guía se refieren a programas más complejos que los que está escribiendo en este momento. A medida que aprenda estructuras de Python más complejas, compartiré las partes relevantes de las pautas de PEP 8.

Inténtalo tú mismo

4-14. PEP 8: consulte la guía de estilo original de PEP 8 en https://python.org/dev/peps/pep-0008/. No usará mucho ahora, pero podría ser interesante hojearlo.

4-15. Revisión de código: Elija tres de los programas que ha escrito en este capítulo y modifique cada uno para cumplir con PEP 8:

- Utilice cuatro espacios para cada nivel de sangría. Configure su editor de texto para insertar cuatro espacios cada vez que presione tabulador, si aún no lo ha hecho (consulte el Apéndice B para obtener instrucciones sobre cómo hacerlo).
- Utilice menos de 80 caracteres en cada línea y configure su editor para mostrar un guía vertical en la posición del carácter 80.
- No utilice líneas en blanco en exceso en sus archivos de programa.

Resumen

En este capítulo aprendió cómo trabajar eficientemente con los elementos de una lista. Aprendió cómo trabajar en una lista usando un bucle for , cómo Python usa la sangría para estructurar un programa y cómo evitar algunos errores de sangría comunes. Aprendiste a hacer listas numéricas simples, así como algunas operaciones que puedes realizar en listas numéricas. Aprendió cómo dividir una lista para trabajar con un subconjunto de elementos y cómo copiar listas correctamente usando una división. También aprendió sobre las tuplas, que brindan un grado de protección a un conjunto de valores que no deberían cambiar, y cómo diseñar su código cada vez más complejo para que sea fácil de leer.

En el Capítulo 5, aprenderá a responder apropiadamente a diferentes condiciones usando sentencias if . Aprenderá a encadenar conjuntos relativamente complejos de pruebas condicionales para responder adecuadamente al tipo exacto de situación o información que está buscando. También aprenderá a usar if declaraciones mientras recorre una lista para realizar acciones específicas con elementos seleccionados de una lista.

5

si declaración s

La programación a menudo implica examinar un conjunto de condiciones y decidir qué acción tomar en función de esas condiciones.

La declaración if de Python le permite examinar el estado actual de un programa y responder adecuadamente a ese estado.

En este capítulo, aprenderá a escribir pruebas condicionales, que le permiten verificar cualquier condición de interés. Aprenderá a escribir declaraciones if simples y aprenderá a crear una serie más compleja de declaraciones if para identificar cuándo están presentes las condiciones exactas que desea. Luego, aplicará este concepto a las listas, de modo que podrá escribir un bucle for que maneje la mayoría de los elementos de una lista de una manera, pero que maneje ciertos elementos con valores específicos de otra manera.

Un ejemplo sencillo

El siguiente breve ejemplo muestra cómo las pruebas le permiten responder correctamente a situaciones especiales. Imagina que tienes una lista de autos y quieres imprimir el nombre de cada auto. Los nombres de los automóviles son nombres propios, por lo que los nombres de la mayoría de los automóviles deben escribirse en mayúsculas y minúsculas. Sin embargo, el valor 'bmw' debe imprimirse en mayúsculas. El siguiente código recorre una lista de nombres de automóviles y busca el valor 'bmw'. Cada vez que el valor es 'bmw', se imprime en mayúsculas en lugar de en mayúsculas y minúsculas:

autos.py

```
coches = ['audi','bmw','subaru','toyota']

para coche en coches:
u si coche == 'bmw':
        imprimir (coche. superior ())
    demás:
        imprimir (coche.título())
```

El bucle de este ejemplo comprueba primero si el valor actual de car es 'bmw' u. Si es así, el valor se imprime en mayúsculas. Si el valor del automóvil es diferente a 'bmw', se imprime en mayúsculas y minúsculas:

```
Audi
BMW
subaru
Toyota
```

Este ejemplo combina varios de los conceptos que aprenderá en este capítulo. Comencemos mirando los tipos de pruebas que puede usar para examinar las condiciones en su programa.

Pruebas Condicionales

En el corazón de cada declaración if hay una expresión que se puede evaluar como verdadera o falsa y se llama *prueba condicional.* Python usa los valores True y False para decidir si se debe ejecutar el código en una declaración if . Si una prueba condicional se evalúa como True, Python ejecuta el código que sigue al if declaración. Si la prueba se evalúa como Falso, Python ignora el código que sigue a la instrucción if .

Comprobación de la igualdad

La mayoría de las pruebas condicionales comparan el valor actual de una variable con un valor específico de interés. La prueba condicional más simple verifica si el valor de una variable es igual al valor de interés:

```
u >>> coche = 'bmw'
en >>> coche == 'bmw'
Verdadero
```

La línea en u establece el valor del automóvil en 'bmw' usando un solo signo igual, como ya ha visto muchas veces. La línea en v verifica si el valor del automóvil es 'bmw' usando un signo igual doble (==). Este *operador de igualdad* devuelve True si los valores del lado izquierdo y derecho del operador coinciden y False si no coinciden. Los valores de este ejemplo coinciden, por lo que Python devuelve True.

Cuando el valor del automóvil es diferente a 'bmw', esta prueba devuelve Falso:

```
u >>> coche = 'audi'
en >>> coche == 'bmw'
    Falso
```

Un solo signo igual es realmente una declaración; podrías leer el código en ti como "Establecer el valor del coche igual a 'audi'". Por otro lado, un doble signo igual, como el que está en v, hace una pregunta: "¿Es el valor del automóvil igual a 'bmw'?" La mayoría de los lenguajes de programación usan signos iguales de esta manera.

Ignorar el caso al verificar la igualdad

La prueba de igualdad distingue entre mayúsculas y minúsculas en Python. Por ejemplo, dos valores con diferente capitalización no se consideran iguales:

```
>>> coche = 'Audi'
>>> coche == 'audi'
Falso
```

Si el caso importa, este comportamiento es ventajoso. Pero si el caso no importa y en su lugar solo desea probar el valor de una variable, puede convertir el valor de la variable a minúsculas antes de hacer la comparación:

```
>>> coche = 'Audi'
>>> coche.inferior() == 'audi'
Verdadero
```

Esta prueba devolvería True sin importar cómo esté formateado el valor 'Audi' porque la prueba ahora no distingue entre mayúsculas y minúsculas. La función lower() no cambia el valor que se almacenó originalmente en car, por lo que puede hacer este tipo de comparación sin afectar la variable original:

```
u >>> coche = 'Audi'
v >>> auto.inferior() == 'audi'
    Verdadero
w >>> coche
    'Audi'
```

En u asignamos la cadena en mayúscula 'Audi' a la variable coche. en v convertimos el valor de car a minúsculas y comparamos el valor en minúsculas

a la cadena 'audi'. Las dos cadenas coinciden, por lo que Python devuelve True. en w podemos ver que el valor almacenado en car no se ha visto afectado por lower() método.

Los sitios web imponen ciertas reglas para los datos que los usuarios ingresan de una manera similar a esta. Por ejemplo, un sitio podría usar una prueba condicional como esta para asegurarse de que cada usuario tenga un nombre de usuario realmente único, no solo una variación de las mayúsculas del nombre de usuario de otra persona. Cuando alguien envía un nuevo nombre de usuario, ese nuevo nombre de usuario se convierte a minúsculas y se compara con las versiones en minúsculas de todos los nombres de usuario existentes. Durante esta verificación, un nombre de usuario como 'John' será rechazado si hay alguna variación de 'john' Esta en uso.

Comprobación de la desigualdad

Cuando desee determinar si dos valores no son iguales, puede combinar un signo de exclamación y un signo igual (!=). El signo de exclamación representa *no,* como lo hace en muchos lenguajes de programación.

Usemos otra instrucción if para examinar cómo usar el operador de desigualdad. Guardaremos un ingrediente de pizza solicitado en una variable y luego imprimiremos un mensaje si la persona no pidió anchoas:

toppings.py

```
request_topping = 'champiñones'

u si se solicita_topping != 'anchoas':
    print("¡Sujeta las anchoas!")
```

La línea en u compara el valor de request_topping con el valor 'anchoas'. Si estos dos valores no coinciden, Python devuelve True y ejecuta el código que sigue a la instrucción if . Si los dos valores coinciden, Python devuelve False y no ejecuta el código que sigue a la instrucción if .

Debido a que el valor de request_topping no es 'anchoas', el print() se ejecuta la función:

```
Aguante las anchoas!
```

La mayoría de las expresiones condicionales que escriba probarán la igualdad, pero a veces le resultará más eficiente probar la desigualdad.

Comparaciones numéricas

Probar valores numéricos es bastante sencillo. Por ejemplo, el siguiente código verifica si una persona tiene 18 años:

```
>>> edad = 18
>>> edad == 18
Verdadero
```

También puedes probar para ver si dos números no son iguales. por ejemplo, el siguiente código imprime un mensaje si la respuesta dada no es correcta:

número_mágico.py

```
respuesta = 17
u si responde != 42:
    print("Esa no es la respuesta correcta. Inténtalo de nuevo!")
```

La prueba condicional en u pasa, porque el valor de la respuesta (17) no es igual a 42. Debido a que la prueba pasa, se ejecuta el bloque de código sangrado:

```
Esa no es la respuesta correcta. ¡Inténtalo de nuevo!
```

También puede incluir varias comparaciones matemáticas en sus declaraciones condicionales, como menor que, menor o igual que, mayor que y mayor o igual que:

```
>>> edad = 19
>>> edad < 21
Verdadero
>>> edad <= 21
Verdadero
>>> edad > 21
Falso
>>> edad >= 21
Falso
```

Cada comparación matemática se puede usar como parte de una declaración if , que puede ayudarlo a detectar las condiciones exactas de interés.

Comprobación de múltiples condiciones

Es posible que desee verificar varias condiciones al mismo tiempo. Por ejemplo, a veces es posible que necesite que dos condiciones sean Verdaderas para realizar una acción. Otras veces puede estar satisfecho con que solo una condición sea Verdadera. Las palabras clave and y or pueden ayudarlo en estas situaciones.

Uso de y para verificar múltiples condiciones

Para verificar si dos condiciones son ambas Verdaderas simultáneamente, use la palabra clave y para combinar las dos pruebas condicionales; si pasa cada prueba, la expresión general se evalúa como verdadera. Si alguna de las pruebas falla o si ambas pruebas fallan, la expresión se evalúa como Falso.

Por ejemplo, puede verificar si dos personas tienen más de 21 años usando la siguiente prueba:

```
u >>> edad_0 = 22
  >>> edad_1 = 18
v >>> edad_0 >= 21 y edad_1 >= 21
  Falso
```

```
w >>> edad_1 = 22
>>> edad_0 >= 21 y edad_1 >= 21
Verdadero
```

En u definimos dos edades, edad_0 y edad_1. En v verificamos si ambas edades tienen 21 años o más. La prueba de la izquierda pasa, pero la prueba de la derecha falla, por lo que la expresión condicional general se evalúa como Falsa. En w cambiamos age_1 a 22. El valor de age_1 ahora es mayor que 21, por lo que ambas pruebas individuales pasan, lo que hace que la expresión condicional general se evalúe como True.

Para mejorar la legibilidad, puede usar paréntesis alrededor de las pruebas individuales, pero no son obligatorios. Si usa paréntesis, su prueba se vería así:

```
(edad_0 >= 21) y (edad_1 >= 21)
```

Usar o para comprobar varias condiciones

La palabra clave o le permite verificar múltiples condiciones también, pero pasa cuando pasa una o ambas pruebas individuales. Una expresión o falla solo cuando ambas pruebas individuales fallan.

Consideremos dos edades nuevamente, pero esta vez buscaremos que solo una persona tenga más de 21 años:

```
u >>> edad_0 = 22
>>> edad_1 = 18
v >>> edad_0 >= 21 o edad_1 >= 21

Verdadero w >>>
edad_0 = 18 >>> edad_0 >= 21 o edad_1 >= 21
Falso
```

Comenzamos con dos variables de edad nuevamente en u. Porque la prueba para age_0 en v pasa, la expresión general se evalúa como True. Luego bajamos age_0 a 18. En la prueba en w, ambas pruebas ahora fallan y la expresión general se evalúa como Falsa.

Comprobar si un valor está en una lista

A veces es importante comprobar si una lista contiene un determinado valor antes de realizar una acción. Por ejemplo, es posible que desee verificar si ya existe un nuevo nombre de usuario en una lista de nombres de usuario actuales antes de completar el registro de alguien en un sitio web. En un proyecto de mapeo, es posible que desee verificar si una ubicación enviada ya existe en una lista de ubicaciones conocidas.

Para saber si un valor en particular ya está en una lista, use la tecla palabra . Consideremos un código que podría escribir para una pizzería. Haremos una lista de ingredientes que un cliente ha solicitado para una pizza y luego verificaremos si ciertos ingredientes están en la lista.

```
>>> Request_toppings = ['champiñones', 'cebollas', 'piña']
u >>> 'champiñones' en los ingredientes solicitados
Verdadero
v >>> 'pepperoni' en los ingredientes solicitados
Falso
```

En u y v, la palabra clave in le dice a Python que verifique la existencia de 'champiñones' y 'pepperoni' en la lista de ingredientes solicitados. Esta técnica es bastante poderosa porque puede crear una lista de valores esenciales y luego verificar fácilmente si el valor que está probando coincide con uno de los valores de la lista.

Comprobar si un valor no está en una lista

Otras veces, es importante saber si un valor no aparece en una lista. Puede utilizar la palabra clave no en esta situación. Por ejemplo, considere una lista de usuarios que tienen prohibido comentar en un foro. Puede verificar si un usuario ha sido prohibido antes de permitir que esa persona envíe un comentario:

baneados _usuarios.py

```
baneados_usuarios = ['andrew', 'carolina', 'david']
usuario = 'marie'

u si el usuario no está en usuarios prohibidos:
    print(f"{user.title()}, puede publicar una respuesta si lo desea").
```

La línea en u se lee bastante claramente. Si el valor del usuario no está en la lista banned_users, Python devuelve True y ejecuta la línea con sangría.

El usuario 'marie' no está en la lista de usuarios prohibidos, por lo que ve un mensaje invitándola a publicar una respuesta:

```
Marie, puedes publicar una respuesta si lo deseas.
```

Expresiones booleanas

A medida que aprenda más sobre programación, escuchará el término *expresión booleana* en algún momento. Una expresión booleana es solo otro nombre para una prueba condicional. Un *valor booleano* es verdadero o falso, al igual que el valor de una expresión condicional después de haberla evaluado.

Los valores booleanos a menudo se usan para realizar un seguimiento de ciertas condiciones, como si un juego se está ejecutando o si un usuario puede editar cierto contenido en un sitio web:

```
juego_activo = Verdadero
can_edit = Falso
```

Los valores booleanos proporcionan una forma eficiente de rastrear el estado de un programa o una condición particular que es importante en su programa.

Inténtalo tú mismo

5-1. Pruebas condicionales: Escribe una serie de pruebas condicionales. Imprima una declaración que describa cada prueba y su predicción para los resultados de cada prueba. Su código debería ser algo como esto:

```
coche = 'subaru'
print("¿Es auto == 'subaru'? Predigo Verdadero.")
imprimir (coche == 'subaru')

print("\nEs auto == 'audi'? Predigo Falso.")
imprimir (coche == 'audi')
```

- Observe detenidamente sus resultados y asegúrese de comprender por qué cada línea se evalúa como Verdadero o Falso.
- Crear al menos diez pruebas. Haga que al menos cinco pruebas se evalúen como Verdadero y otras cinco pruebas se evalúan como Falso.

5-2. Más pruebas condicionales: no tiene que limitar el número de pruebas que crea a diez. Si desea probar más comparaciones, escriba más pruebas y agréguelas a conditional_tests.py. Tenga al menos un resultado verdadero y uno falso para cada uno de los siguientes:

- Pruebas de igualdad y desigualdad con cadenas
- Pruebas usando el método lower()
- Pruebas numéricas que involucran igualdad y desigualdad, mayor que y menor que, mayor que o igual a, y menor que o igual a
- Pruebas usando la palabra clave y y la palabra clave o
- Comprobar si un elemento está en una lista
- Comprobar si un elemento no está en una lista

si declaraciones

Cuando comprenda las pruebas condicionales, puede comenzar a escribir sentencias if . Existen varios tipos diferentes de declaraciones if , y su elección de cuál usar depende de la cantidad de condiciones que necesita probar. Viste varios ejemplos de declaraciones if en la discusión sobre pruebas condicionales, pero ahora profundicemos en el tema.

Declaraciones if simples

El tipo más simple de instrucción if tiene una prueba y una acción:

```
si prueba_condicional:
    hacer algo
```

Puede poner cualquier prueba condicional en la primera línea y casi cualquier acción en el bloque sangrado que sigue a la prueba. Si la prueba condicional se evalúa como True, Python ejecuta el código que sigue a la instrucción if . Si la prueba se evalúa como Falso, Python ignora el código que sigue al if declaración.

Digamos que tenemos una variable que representa la edad de una persona y queremos saber si esa persona tiene la edad suficiente para votar. El siguiente código prueba si la persona puede votar:

```
votando.py edad = 19
u si edad >= 18:
v print("¡Tienes la edad suficiente para votar!")
```

En u, Python comprueba si el valor de la edad es mayor que o igual a 18. Lo es, por lo que Python ejecuta la llamada print() sangrada en v:

```
¡Tienes la edad suficiente para votar!
```

La sangría juega el mismo papel en las sentencias if que en los bucles for . Todas las líneas sangradas después de una declaración if se ejecutarán si la prueba pasa, y todo el bloque de líneas sangradas se ignorará si la prueba no pasa.

Puede tener tantas líneas de código como desee en el bloque que sigue a la instrucción if . Agreguemos otra línea de salida si la persona tiene la edad suficiente para votar, preguntando si la persona ya se ha registrado para votar:

```
edad = 19
si edad >= 18:
    print("¡Tienes la edad suficiente para votar!")
    print("¿Ya se registró para votar?")
```

La prueba condicional pasa, y ambas llamadas print() están sangradas, por lo que se imprimen ambas líneas:

```
¡Tienes la edad suficiente para votar!
¿Ya te registraste para votar?
```

Si el valor de la edad es inferior a 18, este programa no produciría resultados.

Declaraciones if-else

A menudo, deseará realizar una acción cuando pase una prueba condicional y una acción diferente en todos los demás casos. La sintaxis if-else de Python lo hace posible. Un bloque if-else es similar a una declaración if simple , pero la declaración else le permite definir una acción o un conjunto de acciones que se ejecutan cuando falla la prueba condicional.

Mostraremos el mismo mensaje que teníamos anteriormente si la persona es mayor suficiente para votar, pero esta vez agregaremos un mensaje para cualquiera que no tenga la edad suficiente para votar:

```
    edad = 17
u si edad >= 18:
        print("¡Tienes la edad suficiente para votar!")
        print("¿Ya se registró para votar?")
v más:
        print("Lo siento, eres demasiado joven para votar")
        print("¡Por favor regístrese para votar tan pronto como cumpla 18 años!")
```

Si la prueba condicional en u pasa, el primer bloque de print() sangrado se ejecutan las llamadas. Si la prueba se evalúa como Falso, se ejecuta el bloque else en v. Debido a que la edad es menor de 18 años esta vez, la prueba condicional falla y se ejecuta el código en el bloque else :

```
Lo siento, eres demasiado joven para votar.
¡Por favor regístrese para votar tan pronto como cumpla 18 años!
```

Este código funciona porque solo tiene dos situaciones posibles para evaluar: una persona tiene la edad suficiente para votar o no tiene la edad suficiente para votar. El si-más La estructura funciona bien en situaciones en las que desea que Python siempre ejecute una de dos acciones posibles. En una cadena if-else simple como esta, siempre se ejecutará una de las dos acciones.

La cadena if-elif-else

A menudo, deberá probar más de dos situaciones posibles y, para evaluarlas, puede usar la sintaxis if-elif-else de Python. Python ejecuta solo un bloque en una cadena if-elif-else . Ejecuta cada prueba condicional en orden hasta que pasa una. Cuando pasa una prueba, se ejecuta el código que sigue a esa prueba y Python omite el resto de las pruebas.

Muchas situaciones del mundo real involucran más de dos condiciones posibles. Por ejemplo, considere un parque de diversiones que cobra diferentes tarifas para diferentes grupos de edad:

- La entrada para cualquier persona menor de 4 años es gratuita.
- La entrada para cualquier persona entre las edades de 4 y 18 años es de $25.
- La entrada para cualquier persona mayor de 18 años es de $40.

¿Cómo podemos usar una declaración if para determinar la tasa de admisión de una persona? El siguiente código prueba el grupo de edad de una persona y luego imprime un mensaje de precio de admisión:

```
edad de diversión = 12
  _parque.py
u si edad < 4:
        print("El costo de su entrada es $0.")
```

```
v elif edad < 18:
        print("El costo de su entrada es de $25.")
en otro:
        print("El costo de su entrada es de $40.")
```

La prueba if at u prueba si una persona tiene menos de 4 años. Si pasa la prueba, se imprime un mensaje apropiado y Python omite el resto de las pruebas. La línea elif en v es realmente otra prueba if , que se ejecuta solo si la prueba anterior falló. En este punto de la cadena, sabemos que la persona tiene al menos 4 años porque la primera prueba falló. Si la persona es menor de 18 años, se imprime un mensaje apropiado y Python omite el bloque else . Si tanto el si
y las pruebas elif fallan, Python ejecuta el código en el bloque else en w.

En este ejemplo, la prueba en u se evalúa como Falso, por lo que su bloque de código no se ejecuta. Sin embargo, la segunda prueba se evalúa como Verdadero (12 es menor que 18) por lo que se ejecuta su código. El resultado es una oración, informando al usuario del costo de la entrada:

```
Su costo de admisión es de $25.
```

Cualquier edad mayor de 17 años haría que las dos primeras pruebas fallaran. En estas situaciones, se ejecutaría el bloque else y el precio de admisión sería de $40.

En lugar de imprimir el precio de la entrada dentro del bloque if-elif-else , sería más conciso establecer solo el precio dentro de la cadena if-elif-else y luego tener una simple llamada print() que se ejecute después de que se haya evaluado la cadena:

```
edad = 12

si edad < 4:
tu precio = 0
edad < 18:
v precio = 25
demás:
w precio = 40

x print(f"El costo de su entrada es ${precio}").
```

Las líneas en u, v y w establecen el valor del precio según la edad de la persona, como en el ejemplo anterior. Después de que la cadena if-elif-else establece el precio, una llamada print() separada sin sangría ÿ usa este valor para mostrar un mensaje que informa el precio de admisión de la persona.

Este código produce el mismo resultado que el ejemplo anterior, pero el propósito de la cadena if-elif-else es más limitado. En lugar de determinar un precio y mostrar un mensaje, simplemente determina el precio de admisión.
Además de ser más eficiente, este código revisado es más fácil de modificar que el enfoque original. Para cambiar el texto del mensaje de salida, necesitaría cambiar solo una llamada print() en lugar de tres llamadas print() separadas .

Uso de múltiples bloques elif

Puede usar tantos bloques elif en su código como desee. Por ejemplo, si el parque de diversiones implementara un descuento para personas mayores, podría agregar una prueba condicional más al código para determinar si alguien calificó para el descuento para personas mayores. Digamos que cualquier persona mayor de 65 años paga la mitad de la entrada regular, o $20:

```
edad = 12

si edad < 4:
        precio = 0
edad < 18:
        precio = 25
u elif edad < 65:
        precio = 40
v más:
        precio = 20

print(f"El costo de su entrada es ${precio}").
```

La mayor parte de este código no ha cambiado. El segundo bloque elif en u ahora comprueba para asegurarse de que una persona tenga menos de 65 años antes de asignarle la tarifa de admisión total de $40. Observe que el valor asignado en el bloque else en v debe cambiarse a $ 20, porque las únicas edades que llegan a este bloque son las personas de 65 años o más.

Omitir el bloque else

Python no requiere un bloque else al final de una cadena if-elif . Algunas veces un bloque else es útil; a veces es más claro usar una declaración elif adicional que capte la condición específica de interés:

```
edad = 12

si edad < 4:
        precio = 0
edad < 18:
        precio = 25
edad < 65:
        precio = 40
u elif edad >= 65:
        precio = 20

print(f"El costo de su entrada es ${precio}").
```

El bloque extra elif en u asigna un precio de $20 cuando la persona tiene 65 años o más, que es un poco más claro que el bloque general else . Con este cambio, cada bloque de código debe pasar una prueba específica para poder ejecutarse.

El bloque else es una declaración general. Coincide con cualquier condición que no haya coincidido con una prueba if o elif específica , y que a veces puede incluir datos no válidos o incluso maliciosos. Si tiene una condición final específica que está probando, considere usar un bloque elif final y omita el bloque else . Como resultado, obtendrá una mayor confianza en que su código se ejecutará solo en las condiciones correctas.

Prueba de múltiples condiciones

La cadena if-elif-else es poderosa, pero solo es apropiada para usar cuando solo necesita pasar una prueba. Tan pronto como Python encuentra una prueba que pasa, salta el resto de las pruebas. Este comportamiento es beneficioso porque es eficiente y le permite probar una condición específica.

Sin embargo, a veces es importante verificar todas las condiciones de interés. En este caso, debe usar una serie de declaraciones if simples sin bloques elif o else . Esta técnica tiene sentido cuando más de una condición puede ser Verdadera y desea actuar en cada condición que sea Verdadera.

Reconsideremos el ejemplo de la pizzería. Si alguien pide dos ingredientes pizza, deberá asegurarse de incluir ambos ingredientes en su pizza:

```
toppings.py u request_toppings = ['champiñones', 'queso extra']

v si 'champiñones' en los ingredientes solicitados:
        print("Añadiendo champiñones")
w si 'pepperoni' en los ingredientes solicitados:
        print("Añadiendo pepperoni")
x si 'queso extra' en los ingredientes solicitados:
        print("Añadiendo queso extra")

print("\nTerminé de hacer tu pizza!")
```

Comenzamos en u con una lista que contiene los ingredientes solicitados. el si declaración en v comprueba si la persona pidió champiñones en su pizza. Si es así, se imprime un mensaje que confirma ese topping. La prueba para pepperoni en w es otra declaración if simple , no un elif o bien declaración, por lo que esta prueba se ejecuta independientemente de si la prueba anterior pasó o no. El código en x verifica si se solicitó queso adicional independientemente de los resultados de las dos primeras pruebas. Estas tres pruebas independientes se ejecutan cada vez que se ejecuta este programa.

Debido a que se evalúan todas las condiciones de este ejemplo, se agregan champiñones y queso adicional a la pizza:

```
Adición de champiñones.
Agregar queso extra.

¡Terminaste de hacer tu pizza!
```

Este código no funcionaría correctamente si usáramos un bloque if-elif-else , porque el código dejaría de ejecutarse después de que solo pasara una prueba. Así es como se vería:

```
request_toppings = ['champiñones', 'queso extra']

si 'champiñones' en los ingredientes solicitados:
    print("Añadiendo champiñones")
elif 'pepperoni' en los ingredientes solicitados:
    print("Añadiendo pepperoni")
elif 'queso extra' en los ingredientes solicitados:
    print("Añadiendo queso extra")

print("\nTerminé de hacer tu pizza!")
```

La prueba de 'hongos' es la primera prueba que se pasa, por lo que se agregan hongos a la pizza Sin embargo, los valores 'extra cheese' y 'pepperoni' nunca se verifican, porque Python no ejecuta ninguna prueba más allá de la primera prueba que pasa en una cadena if-elif-else . Se agregará el primer aderezo del cliente, pero se perderán todos los demás aderezos:

```
Adición de champiñones.

¡Terminaste de hacer tu pizza!
```

En resumen, si desea que solo se ejecute un bloque de código, use un if-elif otra cadena. Si necesita ejecutar más de un bloque de código, use una serie de declaraciones if independientes .

Inténtalo tú mismo

5-3. Alien Colors #1: Imagina que un alienígena acaba de ser derribado en un juego. Cree una variable llamada alien_color y asígnele un valor de 'verde', 'amarillo' o 'rojo'.

- Escribe un enunciado if para comprobar si el color del extraterrestre es verde. Si es así, imprime un mensaje de que el jugador acaba de ganar 5 puntos.
- Escriba una versión de este programa que pase la prueba if y otra que falle. (La versión que falla no tendrá salida).

5-4. Alien Colors #2: Elija un color para un alienígena como lo hizo en el Ejercicio 5-3, y escriba una cadena if-else .

- Si el color del alienígena es verde, imprime una declaración de que el jugador acaba de ganar 5 puntos por dispararle al alienígena.
- Si el color del alienígena no es verde, imprima una declaración de que el jugador acaba de ganar 10 puntos.
- Escriba una versión de este programa que ejecute el bloque if y otra que ejecute el bloque else .

5-5. Alien Colors #3: Convierte tu cadena if-else del Ejercicio 5-4 en una cadena if-elif else .

- Si el alienígena es verde, imprime un mensaje de que el jugador ganó 5 puntos.
- Si el alienígena es amarillo, imprime un mensaje de que el jugador ganó 10 puntos.
- Si el alienígena es rojo, imprime un mensaje de que el jugador ganó 15 puntos.
- Escriba tres versiones de este programa, asegurándose de que cada mensaje se imprima para el extranjero de color apropiado.

5-6. Etapas de la vida: escriba una cadena if-elif-else que determine la etapa de la vida de una persona. Establezca un valor para la variable edad y luego:

- Si la persona tiene menos de 2 años, imprima un mensaje de que la persona es un bebé.
- Si la persona tiene por lo menos 2 años pero menos de 4, imprima un mensaje que la persona es un niño pequeño.
- Si la persona tiene al menos 4 años pero menos de 13, imprima un mensaje de que la persona es un niño.
- Si la persona tiene por lo menos 13 años pero menos de 20, imprima un mensaje que la persona es un adolescente.
- Si la persona tiene al menos 20 años pero menos de 65, imprima un mensaje que la persona es un adulto.
- Si la persona tiene 65 años o más, imprima un mensaje que indique que la persona es un mayor.

5-7. Fruta favorita: haga una lista de sus frutas favoritas y luego escriba una serie de declaraciones if independientes que verifiquen ciertas frutas en su lista.

- Haz una lista de tus tres frutas favoritas y llámala frutas_favoritas.
- Escriba cinco enunciados if . Cada uno debe verificar si un determinado tipo de fruta está en su lista. Si la fruta está en su lista, el bloque if debe imprimir una declaración, como ¡Realmente le gustan los plátanos!

Uso de sentencias if con listas

Puedes hacer un trabajo interesante cuando combinas listas y sentencias if . Puede buscar valores especiales que deben tratarse de manera diferente a otros valores en la lista. Puede administrar las condiciones cambiantes de manera eficiente, como la disponibilidad de ciertos artículos en un restaurante durante un turno. También puede comenzar a probar que su código funciona como espera en todas las situaciones posibles.

Comprobación de artículos especiales

Este capítulo comenzó con un ejemplo simple que mostraba cómo manejar un valor especial como 'bmw', que necesitaba imprimirse en un formato diferente al de otros valores en la lista. Ahora que tiene una comprensión básica de las pruebas condicionales y las declaraciones if , echemos un vistazo más de cerca a cómo puede observar valores especiales en una lista y manejar esos valores de manera adecuada.

Sigamos con el ejemplo de la pizzería. La pizzería muestra un mensaje cada vez que se agrega un aderezo a su pizza, mientras se prepara. El código para esta acción se puede escribir de manera muy eficiente haciendo una lista de ingredientes que el cliente ha solicitado y usando un bucle para anunciar cada ingrediente a medida que se agrega a la pizza:

toppings.py

```
request_toppings = ['champiñones', 'pimientos verdes', 'queso extra']

para los ingredientes_solicitados en los ingredientes_solicitados:
    print(f"Agregando {requested_topping}.")

print("\nTerminé de hacer tu pizza!")
```

El resultado es sencillo porque este código es solo un bucle for simple:

```
Adición de champiñones.
Agregar pimientos verdes.
Agregar queso extra.

¡Terminaste de hacer tu pizza!
```

Pero, ¿y si la pizzería se queda sin pimientos verdes? Una declaración si dentro del ciclo for puede manejar esta situación apropiadamente:

```
request_toppings = ['champiñones', 'pimientos verdes', 'queso extra']

para los ingredientes_solicitados en los ingredientes_solicitados:
u if request_topping == 'pimientos verdes':
        print("Lo siento, no tenemos pimientos verdes en este momento.")
v más:
        print(f"Agregando {requested_topping}.")

print("\nTerminé de hacer tu pizza!")
```

Esta vez revisamos cada artículo solicitado antes de agregarlo a la pizza. El código en u verifica si la persona solicitó pimientos verdes. Si es así, mostramos un mensaje informándoles por qué no pueden tener pimientos verdes. El bloque else en v asegura que todos los demás ingredientes se agregarán a la pizza.

El resultado muestra que cada aderezo solicitado se maneja de manera adecuada.

```
Adición de champiñones.
Lo sentimos, no tenemos pimientos verdes en este momento.
Agregar queso extra.

¡Terminaste de hacer tu pizza!
```

Comprobar que una lista no está vacía

Hemos hecho una suposición simple sobre cada lista con la que hemos trabajado hasta ahora; hemos asumido que cada lista tiene al menos un elemento en ella. Pronto permitiremos que los usuarios proporcionen la información que está almacenada en una lista, por lo que no podremos asumir que una lista tiene elementos cada vez que se ejecuta un bucle. En esta situación, es útil comprobar si una lista está vacía antes de ejecutar un bucle for .

Como ejemplo, verifiquemos si la lista de ingredientes solicitados está vacía antes de construir la pizza. Si la lista está vacía, le preguntaremos al usuario y nos aseguraremos de que quiera una pizza normal. Si la lista no está vacía, construiremos la pizza tal como lo hicimos en los ejemplos anteriores:

```
has pedido_ingredientes = []

v si se solicitan_toppings:
    para los ingredientes_solicitados en los ingredientes_solicitados:
        print(f"Agregando {requested_topping}.")
    print("\nTerminé de hacer tu pizza!")
en otro:
    print("¿Estás seguro de que quieres una pizza normal?")
```

Esta vez comenzamos con una lista vacía de ingredientes solicitados en u. En lugar de saltar directamente a un bucle for , hacemos una comprobación rápida en v. Cuando se usa el nombre de una lista en una sentencia if , Python devuelve True si la lista contiene al menos un elemento; una lista vacía se evalúa como Falso. Si se solicita_ingredientes pasa la prueba condicional, ejecutamos el mismo bucle for que usamos en el ejemplo anterior. Si la prueba condicional falla, imprimimos un mensaje preguntando al cliente si realmente quiere una pizza sencilla sin ingredientes w.

La lista está vacía en este caso, por lo que el resultado pregunta si el usuario realmente quiere una pizza simple:

```
¿Estás seguro de que quieres una pizza normal?
```

Si la lista no está vacía, la salida mostrará cada ingrediente solicitado se añade a la pizza.

Uso de listas múltiples

La gente pedirá casi cualquier cosa, especialmente cuando se trata de ingredientes para pizza. ¿Qué pasa si un cliente realmente quiere papas fritas en su pizza? Puede usar listas y declaraciones if para asegurarse de que su entrada tenga sentido antes de actuar en consecuencia.

Esté atento a las solicitudes de ingredientes inusuales antes de preparar una pizza.
El siguiente ejemplo define dos listas. La primera es una lista de tops disponibles en la pizzería y la segunda es la lista de toppings que ha solicitado el usuario. Esta vez, cada elemento de los ingredientes solicitados se compara con la lista de ingredientes disponibles antes de agregarlo a la pizza:

```
u available_toppings = ['champiñones', 'aceitunas', 'pimientos verdes',
                        'pepperoni', 'piña', 'queso extra']

v request_toppings = ['champiñones', 'papas fritas', 'queso extra']

w para los ingredientes solicitados en los ingredientes solicitados:
x si se solicita_topping en available_toppings:
        print(f"Agregando {requested_topping}.")
y más:
        print(f"Lo sentimos, no tenemos {requested_topping}").

print("\nTerminé de hacer tu pizza!")
```

En u definimos una lista de ingredientes disponibles en esta pizzería. Tenga en cuenta que esto podría ser una tupla si la pizzería tiene una selección estable de ingredientes. En v, hacemos una lista de toppings que ha pedido un cliente. Tenga en cuenta la solicitud inusual, 'papas fritas'. En w recorremos la lista de ingredientes solicitados.
Dentro del ciclo, primero verificamos si cada ingrediente solicitado está realmente en la lista de ingredientes disponibles x. Si es así, añadimos ese topping a la pizza.
Si el ingrediente solicitado no está en la lista de ingredientes disponibles, el bloque else ejecutará y. El bloque else imprime un mensaje que le dice al usuario qué ingredientes no están disponibles.

Esta sintaxis de código produce resultados limpios e informativos:

```
Adición de champiñones.
Lo siento, no tenemos papas fritas.
Agregar queso extra.

¡Terminaste de hacer tu pizza!
```

¡En solo unas pocas líneas de código, hemos manejado una situación del mundo real de manera bastante efectiva!

Inténtalo tú mismo

5-8. Hola administrador: haga una lista de cinco o más nombres de usuario, incluido el nombre "administrador". Imagine que está escribiendo un código que imprimirá un saludo para cada usuario después de que inicie sesión en un sitio web. Recorra la lista e imprima un saludo para cada usuario:

- Si el nombre de usuario es 'admin', imprima un saludo especial, como Hello admin, ¿Le gustaría ver un informe de estado?
- De lo contrario, imprima un saludo genérico, como Hola Jaden, gracias por iniciar sesión nuevamente.

5-9. Sin usuarios: agregue una prueba if a hello_admin.py para asegurarse de que la lista de usuarios no esté vacía.

- Si la lista está vacía, imprima el mensaje ¡Necesitamos encontrar algunos usuarios!
- Elimine todos los nombres de usuario de su lista y asegúrese de que se imprima el mensaje correcto.

5-10. Comprobación de nombres de usuario: haga lo siguiente para crear un programa que simule cómo los sitios web se aseguran de que todos tengan un nombre de usuario único.

- Haga una lista de cinco o más nombres de usuario llamados current_users.
- Haga otra lista de cinco nombres de usuario llamada new_users. Asegúrese de que uno o dos de los nuevos nombres de usuario también estén en la lista de usuarios_actuales .
- Recorra la lista new_users para ver si ya se ha utilizado cada nuevo nombre de usuario. Si es así, imprima un mensaje que indique que la persona deberá ingresar un nuevo nombre de usuario. Si no se ha utilizado un nombre de usuario, imprima un mensaje que diga que el nombre de usuario está disponible.
- Asegúrese de que su comparación no distinga entre mayúsculas y minúsculas. Si se ha utilizado 'Juan' , No se debe aceptar 'JOHN' . (Para hacer esto, deberá hacer una copia de current_users que contenga las versiones en minúsculas de todos los usuarios existentes).

5-11. Números ordinales: Los números ordinales indican su posición en una lista, como 1º o 2º. La mayoría de los números ordinales terminan en th, excepto 1, 2 y 3.

- Guarde los números del 1 al 9 en una lista.
- Recorra la lista.
- Use una cadena if-elif-else dentro del ciclo para imprimir el final ordinal apropiado para cada número. Su salida debe decir "1st 2nd 3rd 4th 5th 6th 7th 8th 9th", y cada resultado debe estar en una línea separada.

Diseñar sus declaraciones if

En cada ejemplo de este capítulo, ha visto buenos hábitos de estilo. La única recomendación que proporciona PEP 8 para diseñar pruebas condicionales es usar un solo espacio alrededor de los operadores de comparación, como ==, >=, <=. Por ejemplo:

```
si edad < 4:
```

es mejor que:

```
si la edad <4:
```

Dicho espacio no afecta la forma en que Python interpreta su código; es solo hace que su código sea más fácil de leer para usted y para otros.

Inténtalo tú mismo

5-12. Aplicar estilo a las sentencias **if :** revise los programas que escribió en este capítulo y asegúrese de aplicar el estilo adecuado a sus pruebas condicionales.

5-13. Tus ideas: en este punto, eres un programador más capaz que cuando comenzaste este libro. Ahora que tiene una mejor idea de cómo se modelan las situaciones del mundo real en los programas, es posible que esté pensando en algunos problemas que podría resolver con sus propios programas. Registre cualquier idea nueva que tenga sobre problemas que podría querer resolver a medida que sus habilidades de programación continúan mejorando. Considere los juegos que quizás desee escribir, los conjuntos de datos que quizás desee explorar y las aplicaciones web que le gustaría crear.

Resumen

En este capítulo, aprendió a escribir pruebas condicionales, que siempre se evalúan como Verdadero o Falso. Aprendiste a escribir sentencias if simples , if-else cadenas y cadenas if-elif-else . Comenzó a usar estas estructuras para identificar condiciones particulares que necesitaba probar y saber cuándo se cumplieron esas condiciones en sus programas. Aprendió a manejar ciertos elementos en una lista de manera diferente a todos los demás elementos mientras continúa utilizando la eficiencia de un bucle for . También revisó las recomendaciones de estilo de Python para asegurarse de que sus programas cada vez más complejos sigan siendo relativamente fáciles de leer y comprender.

En el Capítulo 6 aprenderá sobre los diccionarios de Python. Un diccionario es similar a una lista, pero le permite conectar piezas de información. Aprenderá a construir diccionarios, recorrerlos y usarlos en combinación con listas y sentencias if . Aprender sobre diccionarios le permitirá modelar una variedad aún más amplia de situaciones del mundo real.



Diccionarios

En este capítulo, aprenderá a usar los diccionarios de Python, que le permiten conectar piezas de información relacionada. lo harás aprenda a acceder a la información una vez que esté en un diccionario y cómo modificar esa información. Debido a que los diccionarios pueden almacenar una cantidad casi ilimitada de información, le mostraré cómo recorrer los datos en un diccionario. Además, aprenderá a anidar diccionarios dentro de listas, listas dentro de diccionarios e incluso diccionarios dentro de otros diccionarios.

Comprender los diccionarios le permite modelar una variedad de situaciones del mundo real. objetos con mayor precisión. Podrá crear un diccionario que represente a una persona y luego almacenar toda la información que desee sobre esa persona. Puede almacenar su nombre, edad, ubicación, profesión y cualquier otro aspecto de una persona que pueda describir. Podrá almacenar dos tipos de

información que se puede unir, como una lista de palabras y sus significados, una lista de nombres de personas y sus números favoritos, una lista de montañas y sus elevaciones, etc.

Un diccionario simple

Considere un juego con extraterrestres que pueden tener diferentes colores y valores de puntos. Este diccionario simple almacena información sobre un extraterrestre en particular:

alien.py

```
alien_0 = {'color': 'verde', 'puntos': 5}

imprimir (alien_0['color'])
imprimir(alien_0['puntos'])
```

El diccionario alien_0 almacena el color y el valor en puntos del alienígena. El último dos líneas acceden y muestran esa información, como se muestra aquí:

```
verde
5
```

Como ocurre con la mayoría de los nuevos conceptos de programación, el uso de diccionarios requiere práctica. Una vez que haya trabajado un poco con los diccionarios, pronto verá la eficacia con la que pueden modelar situaciones del mundo real.

Trabajar con diccionarios

Un *diccionario* en Python es una colección de *pares clave-valor.* Cada *clave* está conectada a un valor y puede usar una clave para acceder al valor asociado con esa clave. El valor de una clave puede ser un número, una cadena, una lista o incluso otro diccionario. De hecho, puede usar cualquier objeto que pueda crear en Python como un valor en un diccionario.

En Python, un diccionario está entre llaves, {}, con una serie de claves pares de valores dentro de las llaves, como se muestra en el ejemplo anterior:

```
alien_0 = {'color': 'verde', 'puntos': 5}
```

Un *par clave-valor* es un conjunto de valores asociados entre sí. Cuando proporciona una clave, Python devuelve el valor asociado con esa clave. Cada clave está conectada a su valor por dos puntos, y los pares clave-valor individuales están separados por comas. Puede almacenar tantos pares clave-valor como desee en un diccionario.

El diccionario más simple tiene exactamente un par clave-valor, como se muestra en esta versión modificada del diccionario alien_0 :

```
alien_0 = {'color': 'verde'}
```

Este diccionario almacena una pieza de información sobre alien_0, a saber, el color del alienígena. La cadena 'color' es una clave en este diccionario y su valor asociado es 'verde'.

Acceso a valores en un diccionario

Para obtener el valor asociado con una clave, ingrese el nombre del diccionario y luego coloque la clave dentro de un conjunto de corchetes, como se muestra aquí:

alien.py
```
alien_0 = {'color': 'verde'}
imprimir (alien_0['color'])
```

Esto devuelve el valor asociado con la clave 'color' del diccionario alien_0:

```
verde
```

Puede tener un número ilimitado de pares clave-valor en un diccionario. Por ejemplo, aquí está el diccionario alien_0 original con dos pares clave-valor:

```
alien_0 = {'color': 'verde', 'puntos': 5}
```

Ahora puede acceder al color o al valor en puntos de alien_0. Si un jugador derriba a este alienígena, puedes buscar cuántos puntos debería ganar usando un código como este:

```
alien_0 = {'color': 'verde', 'puntos': 5}

u nuevos_puntos = alien_0['puntos']
v print(f"¡Acabas de ganar {nuevos_puntos} puntos!")
```

Una vez que se ha definido el diccionario, el código en u extrae el valor asociado con los 'puntos' clave del diccionario. Este valor se asigna luego a la variable new_points. La línea en v imprime una declaración sobre cuántos puntos acaba de ganar el jugador:

```
¡Acabas de ganar 5 puntos!
```

Si ejecuta este código cada vez que se derriba a un alienígena, se recuperará el valor en puntos del alienígena.

Adición de nuevos pares clave-valor

Los diccionarios son estructuras dinámicas y puede agregar nuevos pares clave-valor a un diccionario en cualquier momento. Por ejemplo, para agregar un nuevo par clave-valor, daría el nombre del diccionario seguido de la nueva clave entre corchetes junto con el nuevo valor.

Agreguemos dos nuevos elementos de información al diccionario alien_0 : las coordenadas x e y del alienígena, que nos ayudarán a mostrar el alienígena en un

posición particular en la pantalla. Coloquemos al extraterrestre en el borde izquierdo de la pantalla, 25 píxeles por debajo de la parte superior. Debido a que las coordenadas de la pantalla generalmente comienzan en la esquina superior izquierda de la pantalla, colocaremos el extraterrestre en el borde izquierdo de la pantalla configurando la coordenada x en 0 y 25 píxeles desde la parte superior configurando su coordenada y en positivo 25, como se muestra aquí:

alien.py

```
alien_0 = {'color': 'verde', 'puntos': 5}
imprimir (alien_0)

u alien_0['x_position'] = 0
v alien_0['posición_y'] = 25
imprimir (alien_0)
```

Comenzamos definiendo el mismo diccionario con el que hemos estado trabajando. Luego imprimimos este diccionario, mostrando una instantánea de su información. en ti agregamos un nuevo par clave-valor al diccionario: clave 'x_position' y valor 0. Hacemos lo mismo para la clave 'y_position' en v. Cuando imprimimos el diccionario modificado, vemos los dos pares clave-valor adicionales:

```
{'color': 'verde', 'puntos': 5}
{'color': 'verde', 'puntos': 5, 'posición_y': 25, 'posición_x': 0}
```

La versión final del diccionario contiene cuatro pares clave-valor. Los dos originales especifican el color y el valor del punto, y dos más especifican la posición del alienígena.

A partir de Python 3.7, los diccionarios conservan el orden en que fueron definidos. Cuando imprima un diccionario o recorra sus elementos, verá los elementos en el mismo orden en que se agregaron al diccionario.

Comenzando con un diccionario vacío

A veces es conveniente, o incluso necesario, comenzar con un diccionario vacío y luego agregarle cada elemento nuevo. Para comenzar a llenar un diccionario vacío, defina un diccionario con un conjunto vacío de llaves y luego agregue cada par clave-valor en su propia línea. Por ejemplo, aquí se explica cómo construir el diccionario alien_0 usando este enfoque:

alien.py

```
extranjero_0 = {}

alien_0['color'] = 'verde'
alien_0['puntos'] = 5

imprimir (alien_0)
```

Aquí definimos un diccionario alien_0 vacío y luego le agregamos color y valores de puntos. El resultado es el diccionario que hemos estado usando en ejemplos anteriores:

```
{'color': 'verde', 'puntos': 5}
```

Por lo general, utilizará diccionarios vacíos al almacenar datos proporcionados por el usuario en un diccionario o cuando escribe código que genera una gran cantidad de pares clave-valor automáticamente.

Modificación de valores en un

diccionario Para modificar un valor en un diccionario, ingrese el nombre del diccionario con la clave entre corchetes y luego el nuevo valor que desea asociar con esa clave. Por ejemplo, considere un extraterrestre que cambia de verde a amarillo a medida que avanza el juego:

alien.py

```
alien_0 = {'color': 'verde'}
print(f"El extraterrestre es {alien_0['color']}").

alien_0['color'] = 'amarillo'
print(f"El extraterrestre ahora es {alien_0['color']}").
```

Primero definimos un diccionario para alien_0 que contiene solo el color del alienígena; luego cambiamos el valor asociado a la clave 'color' a 'amarillo'. El resultado muestra que el alienígena ha cambiado de verde a amarillo:

```
El extraterrestre es verde.
El alienígena ahora es amarillo.
```

Para un ejemplo más interesante, sigamos la posición de un extraterrestre que puede moverse a diferentes velocidades. Guardaremos un valor que represente la velocidad actual del alienígena y luego lo usaremos para determinar qué tan lejos a la derecha debe moverse el alienígena:

```
alien_0 = {'posición_x': 0, 'posición_y': 25, 'velocidad': 'media'}
print(f"Posición original: {alien_0['x_position']}")

# Mueve al alienígena a la derecha.
# Determine qué tan lejos mover al alienígena en función de su velocidad actual.
u if alien_0['velocidad'] == 'lento':
    incremento_x = 1
elif alien_0['velocidad'] == 'medio':
    incremento_x = 2
demás:
    # Este debe ser un extraterrestre rapido.
    incremento_x = 3

# La nueva posición es la posición anterior más el incremento.
v alien_0['x_position'] = alien_0['x_position'] + x_increment

print(f"Nueva posición: {alien_0['x_position']}")
```

Comenzamos definiendo un extraterrestre con una posición x inicial y *una* posición *y* , y una velocidad de 'media'. Hemos omitido el color y los valores de puntos para el

En aras de la simplicidad, pero este ejemplo funcionaría de la misma manera si también incluyeras esos pares clave-valor. También imprimimos el valor original de x_position para ver cuánto se mueve el alienígena hacia la derecha.

En u, una cadena if-elif-else determina cuánto debe moverse el alienígena hacia la derecha y asigna este valor a la variable x_increment. Si la velocidad del alienígena es 'lenta', se mueve una unidad hacia la derecha; si la velocidad es 'media', se mueve dos unidades a la derecha; y si es 'rápido', se mueve tres unidades a la derecha. Una vez que se ha calculado el incremento, se suma al valor de x_position en v, y el resultado se almacena en x_position del diccionario.

Debido a que este es un alienígena de velocidad media, su posición se desplaza dos unidades a la derecha:

```
Posición x original: 0
Nueva posición x: 2
```

Esta técnica es genial: al cambiar un valor en el diccionario del alienígena, puedes cambiar el comportamiento general del alienígena. Por ejemplo, para convertir este extraterrestre de velocidad media en un extraterrestre rápido, agregaría la línea:

```
alien_0['velocidad'] = 'rápido'
```

El bloque if-elif-else asignaría un valor mayor a x_increment la próxima vez que se ejecute el código.

Eliminación de pares clave-valor

Cuando ya no necesite una parte de la información almacenada en un diccionario, puede usar la instrucción del para eliminar por completo un par clave-valor. Todo lo que necesita es el nombre del diccionario y la clave que desea retirar.

Por ejemplo, eliminemos la clave 'puntos' del diccionario alien_0 junto con su valor:

alien.py

```
alien_0 = {'color': 'verde', 'puntos': 5}
imprimir (alien_0)

u del alien_0['puntos']
imprimir (alien_0)
```

La línea en u le dice a Python que elimine los 'puntos' clave del diccionario alien_0 y que también elimine el valor asociado con esa clave. El resultado muestra que los 'puntos' clave y su valor de 5 se eliminan del diccionario, pero el resto del diccionario no se ve afectado:

```
{'color': 'verde', 'puntos': 5}
{'color verde'}
```

Tenga en cuenta que el par clave-valor eliminado se elimina de forma permanente.

Un diccionario de objetos similares

El ejemplo anterior implicó almacenar diferentes tipos de información sobre un objeto, un extraterrestre en un juego. También puede usar un diccionario para almacenar un tipo de información sobre muchos objetos. Por ejemplo, supongamos que desea encuestar a varias personas y preguntarles cuál es su lenguaje de programación favorito. Un diccionario es útil para almacenar los resultados de una encuesta simple, como esta:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }
```

Como puede ver, hemos dividido un diccionario más grande en varias líneas. Cada key es el nombre de una persona que respondió a la encuesta, y cada valor es su elección de idioma. Cuando sepa que necesitará más de una línea para definir un diccionario, presione Intro después de la llave de apertura. Luego, sangra la línea siguiente un nivel (cuatro espacios) y escribe el primer par clave-valor, seguido de una coma. A partir de este momento, cuando presione Intro, su editor de texto debería sangrar automáticamente todos los pares clave-valor subsiguientes para que coincidan con el primer par clave-valor.

Una vez que haya terminado de definir el diccionario, agregue una llave de cierre en una nueva línea después del último par clave-valor y sangre un nivel para que se alinee con las claves del diccionario. También es una buena práctica incluir una coma después del último par clave-valor, de modo que esté listo para agregar un nuevo par clave-valor en la línea siguiente.

La mayoría de los editores tienen alguna funcionalidad que lo ayuda a formatear listas extendidas y dic. cionarios de manera similar a este ejemplo. También hay disponibles otras formas aceptables de formatear diccionarios largos, por lo que es posible que vea un formato ligeramente diferente en su editor o en otras fuentes.

Para usar este diccionario, dado el nombre de una persona que realizó la encuesta, puede buscar fácilmente su idioma favorito:

favorito _idiomas.py

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }

u idioma = idiomas_favoritos['sarah'].title()
print(f"El idioma favorito de Sarah es {idioma}").
```

Para ver qué idioma eligió Sarah, solicitamos el valor en:

```
idiomas_favoritos['sarah']
```

Usamos esta sintaxis para extraer el idioma favorito de Sarah del diccionario en u y asignarlo a la variable idioma. Crear una nueva variable aquí hace que la llamada print() sea mucho más limpia . El resultado muestra el idioma favorito de Sarah:

```
El idioma favorito de Sarah es C.
```

Podría usar esta misma sintaxis con cualquier individuo representado en el diccionario.

Usando get() para acceder a los valores

El uso de claves entre corchetes para recuperar el valor que le interesa de un diccionario puede causar un problema potencial: si la clave que solicita no existe, obtendrá un error.

Veamos qué sucede cuando pides el valor en puntos de un extraterrestre que no tiene establecido un valor en puntos:

alien_no_points.py

```
alien_0 = {'color': 'verde', 'velocidad': 'lento'}
imprimir(alien_0['puntos'])
```

Esto da como resultado un rastreo, que muestra un KeyError:

```
Rastreo (llamadas recientes más última):
  Archivo "alien_no_points.py", línea 2, en <módulo>
    imprimir(alien_0['puntos'])
KeyError: 'puntos'
```

Aprenderá más sobre cómo manejar errores como este en general en el Capítulo 10. Para los diccionarios, específicamente, puede usar el método get() para establecer un valor predeterminado que se devolverá si la clave solicitada no existe.

El método get() requiere una clave como primer argumento. Como segundo argumento opcional, puede pasar el valor que se devolverá si la clave no existir:

```
alien_0 = {'color': 'verde', 'velocidad': 'lento'}

point_value = alien_0.get('puntos', 'Sin valor de puntos asignado.')
imprimir (punto_valor)
```

Si los 'puntos' clave existen en el diccionario, obtendrá el valor correspondiente. Si no es así, obtiene el valor predeterminado. En este caso, los puntos no existen y recibimos un mensaje limpio en lugar de un error:

```
No se asigna ningún valor en puntos.
```

Si existe la posibilidad de que la clave que está solicitando no exista, considere utilizando el método get() en lugar de la notación de corchetes.

N ota *omite el segundo argumento en la llamada a get() y la clave no existe, Python devolverá el valor Ninguno. El valor especial Ninguno significa que "no existe ningún valor". Esto no es un error: es un valor especial destinado a indicar la ausencia de un valor. Verá más usos para Ninguno en el Capítulo 8.*

Inténtalo tú mismo

6-1. Persona: use un diccionario para almacenar información sobre una persona que conoce. Almacene su nombre, apellido, edad y la ciudad en la que vive. Debe tener claves como nombre , apellido, edad y ciudad. Imprima cada pieza de información almacenada en su diccionario.

6-2. Números favoritos: use un diccionario para almacenar los números favoritos de las personas. Piensa en cinco nombres y utilízalos como claves en tu diccionario. Piense en un número favorito para cada persona y guárdelo como un valor en su diccionario. Escriba el nombre de cada persona y su número favorito. Para divertirse aún más, haga una encuesta a algunos amigos y obtenga algunos datos reales para su programa.

6-3. Glosario: se puede usar un diccionario de Python para modelar un diccionario real. Sin embargo, para evitar confusiones, llamémoslo glosario.

- Piense en cinco palabras de programación que haya aprendido en los capítulos anteriores. Utilice estas palabras como claves en su glosario y almacene sus significados como valores.

- Imprima cada palabra y su significado como una salida con un formato ordenado. podrías imprima la palabra seguida de dos puntos y luego su significado, o imprima la palabra en una línea y luego imprima su significado con sangría en una segunda línea. Utilice el carácter de nueva línea (\n) para insertar una línea en blanco entre cada par de palabra y significado en su salida.

Bucle a través de un diccionario

Un solo diccionario de Python puede contener solo unos pocos pares clave-valor o millones de pares. Debido a que un diccionario puede contener grandes cantidades de datos, Python le permite recorrer un diccionario. Los diccionarios se pueden utilizar para almacenar información de diversas formas; por lo tanto, existen varias formas diferentes de recorrerlos. Puede recorrer todos los pares clave-valor de un diccionario, sus claves o sus valores.

Bucle a través de todos los pares clave-valor

Antes de explorar los diferentes enfoques de los bucles, consideremos un nuevo diccionario diseñado para almacenar información sobre un usuario en un sitio web. Él

El siguiente diccionario almacenaría el nombre de usuario, el nombre y el apellido de una persona:

```
usuario_0 = {
    'usuario': 'efermi',
    'primero': 'enrico',
    'último': 'fermi',
    }
```

Puede acceder a cualquier información individual sobre user_0 en función de lo que ya aprendió en este capítulo. Pero, ¿y si quisieras ver todo lo almacenado en el diccionario de este usuario? Para hacerlo, puede recorrer el diccionario usando un bucle for :

usuario.py

```
usuario_0 = {
    'usuario': 'efermi',
    'primero': 'enrico',
    'último': 'fermi',
    }

u para clave, valor en user_0.items():
v print(f"\nClave: {clave}")
w imprimir(f"Valor: {valor}")
```

Como se muestra en u, para escribir un bucle for para un diccionario, crea nombres para las dos variables que contendrán la clave y el valor en cada par clave-valor. Puede elegir los nombres que desee para estas dos variables. Este código funcionaría igual de bien si hubiera usado abreviaturas para los nombres de las variables, como esta:

```
para k, v en user_0.items()
```

La segunda mitad de la instrucción for en u incluye el nombre del diccionario seguido del método items(), que devuelve una lista de pares clave-valor. El bucle for luego asigna cada uno de estos pares a las dos variables provistas. En el ejemplo anterior, usamos las variables para imprimir cada clave v, seguida del valor asociado w. El "\n" en la primera llamada print() asegura que se inserte una línea en blanco antes de cada par clave-valor en la salida:

```
Clave: última
valor: fermi

Clave: primero
Valor: Enrique

Clave: nombre de usuario
Valor: efermi
```

Recorrer todos los pares clave-valor funciona particularmente bien para la dicción. arios como el ejemplo de *favorite_languages.py* en la página 97, que almacena el mismo tipo de información para muchas claves diferentes. Si recorre el diccionario de idiomas_favoritos , obtiene el nombre de cada persona en el diccionario y su lenguaje de programación favorito. Debido a que las claves siempre se refieren al nombre de una persona y el valor siempre es un idioma, usaremos las variables nombre e idioma en el bucle en lugar de clave y valor. Esto hará que sea más fácil seguir lo que sucede dentro del ciclo:

favorito
_idiomas.py

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }

u para nombre, idioma en favoritos_idiomas.elementos():
v print(f"{nombre.título()} El idioma favorito de {idioma.título()}.")
```

El código en u le dice a Python que recorra cada par clave-valor en el diccionario. A medida que avanza en cada par, la clave se asigna al nombre de la variable y el valor se asigna al idioma de la variable. Estos nombres descriptivos hacen que sea mucho más fácil ver lo que está haciendo la llamada print() en v.

Ahora, en solo unas pocas líneas de código, podemos mostrar toda la información de la encuesta:

```
El lenguaje favorito de Jen es Python.
El idioma favorito de Sarah es C.
El idioma favorito de Edward es Ruby.
El lenguaje favorito de Phil es Python.
```

Este tipo de bucle funcionaría igual de bien si nuestro diccionario almacenara los resultados de encuestar a mil o incluso a un millón de personas.

Recorriendo todas las claves de un diccionario

El método keys() es útil cuando no necesita trabajar con todos los valores en un diccionario. Recorramos el diccionario de idiomas favoritos e imprimamos los nombres de todos los que respondieron la encuesta:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }

u para el nombre en favoritos_idiomas.keys():
    imprimir (nombre. título ())
```

La línea en u le dice a Python que extraiga todas las claves del diccionario idiomas_favoritos y las asigne una a la vez al nombre de la variable. El resultado muestra los nombres de todos los que respondieron la encuesta:

```
Sólo
Sara
Eduardo
phil
```

Recorrer las teclas es en realidad el comportamiento predeterminado cuando se recorre un diccionario, por lo que este código tendría exactamente el mismo resultado si escribió . . .

```
para nombre en idiomas_favoritos:
```

en vez de . . .

```
para el nombre en favoritos_idiomas.keys():
```

Puede optar por usar el método keys() explícitamente si hace que su código sea más fácil de leer, o puede omitirlo si lo desea.

Puede acceder al valor asociado con cualquier clave que le interese dentro del bucle utilizando la clave actual. Imprimamos un mensaje para un par de amigos sobre los idiomas que eligieron. Recorreremos los nombres en el diccionario como lo hicimos anteriormente, pero cuando el nombre coincida con uno de nuestros amigos, mostraremos un mensaje sobre su idioma favorito:

```
idiomas_favoritos = {
    --recorte--
    }

amigos = ['phil', 'sarah']
para el nombre en favoritos_idiomas.keys():
    imprimir (nombre. título ())

v si nombre en amigos:
en          idioma = idiomas_favoritos[nombre].título()
            print(f"\t{nombre.título()}, ¡veo que amas {idioma}!")
```

En u hacemos una lista de amigos a los que queremos imprimir un mensaje. En el interior el bucle, imprimimos el nombre de cada persona. Luego en v verificamos si el nombre con el que estamos trabajando está en la lista de amigos. Si es así, determinamos el idioma favorito de la persona utilizando el nombre del diccionario y el valor actual de nombre como clave w. Luego imprimimos un saludo especial, que incluye una referencia al idioma de su elección.

El nombre de todos está impreso, pero nuestros amigos reciben un mensaje especial:

```
Hola Jen.
Hola Sarah.
    ¡Sarah, veo que amas a C!
Hola Eduardo.
```

```
hola phil
¡Phil, veo que te encanta Python!
```

También puede usar el método keys() para averiguar si se encuestó a una persona en particular. Esta vez, averigüemos si Erin participó en la encuesta:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }

u si 'erin' no está en favorite_languages.keys():
    print("Erin, ¡haz nuestra encuesta!")
```

El método keys() no es solo para bucles: en realidad devuelve una lista de todos las teclas, y la línea en u simplemente verifica si 'erin' está en esta lista. Como no lo es, se imprime un mensaje invitándola a participar en la encuesta:

```
¡Erin, por favor toma nuestra encuesta!
```

Recorriendo las teclas de un diccionario en un orden particular

A partir de Python 3.7, recorrer un diccionario devuelve los elementos en el mismo orden en que se insertaron. A veces, sin embargo, querrá recorrer un diccionario en un orden diferente.

Una forma de hacer esto es ordenar las claves a medida que se devuelven en el bucle for . Puede usar la función sorted() para obtener una copia de las claves en orden:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }

para el nombre ordenado (favorite_languages.keys()):
    print(f"{nombre.título()}, gracias por participar en la encuesta").
```

Esta declaración for es como otras declaraciones for excepto que hemos envuelto la función sorted() alrededor del método dictionary.keys() . Esto le dice a Python que enumere todas las claves en el diccionario y ordene esa lista antes de recorrerla. El resultado muestra a todos los que respondieron la encuesta, con los nombres mostrados en orden:

```
Edward, gracias por participar en la encuesta.
Jen, gracias por participar en la encuesta.
Phil, gracias por participar en la encuesta.
Sara, gracias por participar en la encuesta.
```

Recorriendo todos los valores en un diccionario

Si está interesado principalmente en los valores que contiene un diccionario, puede usar el método de valores () para devolver una lista de valores sin ninguna clave. Por ejemplo, digamos que simplemente queremos una lista de todos los idiomas elegidos en nuestra encuesta de idiomas de programación sin el nombre de la persona que eligió cada idioma:

```
idiomas_favoritos = {
    'jen': 'pitón',
    'sara': 'c',
    'eduardo': 'rubí',
    'phil': 'pitón'
    }

print("Se han mencionado los siguientes idiomas:")
para idioma en favoritos_idiomas.valores():
    imprimir(idioma.título())
```

La declaración for aquí extrae cada valor del diccionario y lo asigna a la variable idioma. Cuando se imprimen estos valores, obtenemos una lista de todos los idiomas elegidos:

```
Se han mencionado los siguientes idiomas:
Pitón
C
Pitón
Rubí
```

Este enfoque extrae todos los valores del diccionario sin verificar para repeticiones. Eso podría funcionar bien con una pequeña cantidad de valores, pero en una encuesta con una gran cantidad de encuestados, esto daría como resultado una lista muy repetitiva. Para ver cada idioma elegido sin repetición, podemos usar un conjunto. Un *conjunto* es una colección en la que cada elemento debe ser único:

```
idiomas_favoritos = {
    --recorte--
    }

print("Se han mencionado los siguientes idiomas:")
u para idioma en conjunto (favorite_languages.values()):
    imprimir(idioma.título())
```

Cuando ajusta set() alrededor de una lista que contiene elementos duplicados, Python identifica los elementos únicos en la lista y crea un conjunto a partir de esos elementos. en ti usamos set() para extraer los idiomas únicos en favorite_languages.values().

El resultado es una lista no repetitiva de idiomas que se han mencionado por personas que tomaron la encuesta:

```
Se han mencionado los siguientes idiomas:
Pitón
```

```
c
Rubí
```

A medida que continúe aprendiendo sobre Python, a menudo encontrará una característica integrada del lenguaje que lo ayudará a hacer exactamente lo que desea con sus datos.

N ota *uede construir un conjunto directamente usando llaves y separando los elementos con comas:*

```
>>> idiomas = {'python', 'ruby', 'python', 'c'}
>>> idiomas
{'rubí', 'pitón', 'c'
```

Es fácil confundir conjuntos con diccionarios porque ambos están entre llaves. Cuando ve llaves pero no pares clave-valor, probablemente esté viendo un conjunto. A diferencia de las listas y los diccionarios, los conjuntos no conservan elementos en ningún orden específico.

Inténtalo tú mismo

6-4. Glosario 2: ahora que sabe cómo recorrer un diccionario, limpie el código del ejercicio 6-3 (página 99) reemplazando su serie de print() llamadas con un bucle que recorre las claves y los valores del diccionario. Cuando esté seguro de que su ciclo funciona, agregue cinco términos más de Python a su glosario. Cuando vuelva a ejecutar su programa, estas nuevas palabras y significados deberían incluirse automáticamente en la salida.

6-5. Ríos: haga un diccionario que contenga tres ríos principales y el país por el que pasa cada río. Un par clave-valor podría ser 'nilo': 'egipto'.

- Use un bucle para imprimir una oración sobre cada río, como El Nilo atraviesa Egipto.
- Utilice un bucle para imprimir el nombre de cada río incluido en el diccionario.
- Utilice un bucle para imprimir el nombre de cada país incluido en el diccionario.

6-6. Sondeo: use el código en favorite_languages.py (página 97).

- Haga una lista de las personas que deberían participar en la encuesta de idiomas favoritos. Incluye algunos nombres que ya estén en el diccionario y otros que no.
- Recorra la lista de personas que deberían realizar la encuesta. Si ya han realizado la encuesta, imprima un mensaje agradeciéndoles por responder. Si aún no han realizado la encuesta, imprima un mensaje invitándoles a realizar la encuesta.

Anidamiento

A veces querrá almacenar varios diccionarios en una lista o una lista de elementos como valor en un diccionario. Esto se llama *anidamiento.* Puede anidar diccionarios dentro de una lista, una lista de elementos dentro de un diccionario o incluso un diccionario dentro de otro diccionario. El anidamiento es una característica poderosa, como lo demostrarán los siguientes ejemplos.

Una lista de diccionarios

El diccionario alien_0 contiene una variedad de información sobre un extraterrestre, pero no tiene espacio para almacenar información sobre un segundo extraterrestre, y mucho menos una pantalla llena de extraterrestres. ¿Cómo puedes manejar una flota de extraterrestres? Una forma es hacer una lista de extraterrestres en la que cada extraterrestre sea un diccionario de información sobre ese extraterrestre. Por ejemplo, el siguiente código crea una lista de tres extraterrestres:

extraterrestres.py

```
alien_0 = {'color': 'verde', 'puntos': 5}
alien_1 = {'color': 'amarillo', 'puntos': 10}
alien_2 = {'color': 'rojo', 'puntos': 15}

u alienígenas = [alien_0, alien_1, alien_2]

para alienígena en alienígenas:
    imprimir (extranjero)
```

Primero creamos tres diccionarios, cada uno representando un extraterrestre diferente. En u almacenamos cada uno de estos diccionarios en una lista llamada extranjeros. Finalmente, recorremos la lista e imprimimos cada extraterrestre:

```
{'color': 'verde', 'puntos': 5} {'color': 'amarillo',
'puntos': 10} {'color': 'rojo', 'puntos': 15}
```

Un ejemplo más realista involucraría a más de tres alienígenas con un código que genera automáticamente cada alienígena. En el siguiente ejemplo, usamos range() para crear una flota de 30 alienígenas:

```
# Haga una lista vacía para almacenar extraterrestres.
extraterrestres = []

# Haz 30 alienígenas verdes.
u para alien_number en el rango (30):
v new_alien = {'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
w alienígenas.append(nuevo_alienígena)

# Muestra los primeros 5 alienígenas.
x para extranjero en extranjeros [: 5]:
    imprimir (extranjero)
imprimir("...")

# Muestra cuántos extraterrestres se han creado.
y print(f"Número total de extranjeros: {len(extranjeros)}")
```

Este ejemplo comienza con una lista vacía para contener todos los alienígenas que se crearán. En u range() devuelve una serie de números, que solo le dice a Python cuántas veces queremos que se repita el bucle. Cada vez que se ejecuta el bucle, creamos un nuevo alienígena v y luego agregamos cada nuevo alienígena a la lista de alienígenas w. En x usamos un segmento para imprimir los primeros cinco alienígenas, y luego en y imprimimos la longitud de la lista para demostrar que en realidad hemos generado la flota completa de 30 alienígenas:

```
{'velocidad': 'lento', 'color': 'verde', 'puntos': 5} {'velocidad': 'lento',
'color': 'verde', 'puntos': 5} {'velocidad' : 'lento', 'color': 'verde',
'puntos': 5} {'velocidad': 'lento', 'color': 'verde', 'puntos': 5} {'velocidad':
'lento' , 'color': 'verde', 'puntos': 5}

...

Número total de extranjeros: 30
```

Todos estos alienígenas tienen las mismas características, pero Python considera a cada uno un objeto separado, lo que nos permite modificar cada alienígena individualmente.

¿Cómo podrías trabajar con un grupo de extraterrestres como este? Imagina que un aspecto de un juego tiene algunos extraterrestres que cambian de color y se mueven más rápido a medida que avanza el juego. Cuando sea el momento de cambiar los colores, podemos usar un bucle for y una instrucción if para cambiar el color de los alienígenas. Por ejemplo, para cambiar los tres primeros extraterrestres a extraterrestres amarillos de velocidad media que valen 10 puntos cada uno, podríamos hacer esto:

```
# Haga una lista vacía para almacenar extraterrestres.
extraterrestres = []

# Haz 30 alienígenas verdes.
para alien_number en rango (30):
    new_alien = {'color': 'verde', 'puntos': 5, 'velocidad': 'lento'}
    alienígenas.append(nuevo_alienígena)

para extranjero en extranjeros[:3]:
    if alien['color'] == 'verde':
        extranjero['color'] = 'amarillo'
        extranjero['velocidad'] = 'medio'
        extranjero['puntos'] = 10

# Muestra los primeros 5 alienígenas.
para extranjero en extranjeros[:5]:
    imprimir (extranjero)
imprimir("...")
```

Debido a que queremos modificar los primeros tres alienígenas, recorremos un segmento que incluye solo los primeros tres alienígenas. Todos los extraterrestres son verdes ahora, pero ese no será siempre el caso, por lo que escribimos una declaración if para asegurarnos

solo estamos modificando alienígenas verdes. Si el alienígena es verde, cambiamos el color a 'amarillo', la velocidad a 'media' y el valor de puntos a 10, como se muestra en el siguiente resultado:

```
{'velocidad': 'medio', 'color': 'amarillo', 'puntos': 10} {'velocidad':
'medio', 'color': 'amarillo', 'puntos': 10} {'velocidad' : 'medio', 'color':
'amarillo', 'puntos': 10} {'velocidad': 'lento', 'color': 'verde', 'puntos':
5} {'velocidad': 'lento' , 'color': 'verde', 'puntos': 5}

...
```

Puede expandir este ciclo agregando un bloque elif que convierte a los alienígenas amarillos en rojos, que se mueven rápidamente, con un valor de 15 puntos cada uno. Sin volver a mostrar todo el programa, ese ciclo se vería así:

```
para extranjero en extranjeros[0:3]:
    if alien['color'] == 'verde':
        extranjero['color'] = 'amarillo'
        extranjero['velocidad'] = 'medio'
        extranjero['puntos'] = 10
    elif alien['color'] == 'amarillo':
        extranjero['color'] = 'rojo'
        extranjero['velocidad'] = 'rápido'
        extranjero['puntos'] = 15
```

Es común almacenar varios diccionarios en una lista cuando cada dic

tionary contiene muchos tipos de información sobre un objeto. Por ejemplo, puede crear un diccionario para cada usuario en un sitio web, como hicimos en *user.py* en la página 100 y almacene los diccionarios individuales en una lista llamada usuarios. Todos los diccionarios de la lista deben tener una estructura idéntica para que pueda recorrer la lista y trabajar con cada objeto de diccionario de la misma manera.

Una lista en un diccionario

En lugar de poner un diccionario dentro de una lista, a veces es útil poner una lista dentro de un diccionario. Por ejemplo, considere cómo podría describir una pizza que alguien está ordenando. Si tuviera que usar solo una lista, todo lo que realmente podría almacenar es una lista de los ingredientes de la pizza. Con un diccionario, una lista de los mejores pings puede ser solo un aspecto de la pizza que estás describiendo.

En el siguiente ejemplo, se almacenan dos tipos de información para cada pizza: un tipo de corteza y una lista de ingredientes. La lista de toppings es un valor asociado a la clave 'toppings'. Para utilizar los elementos de la lista, le damos el nombre del diccionario y la tecla 'toppings', como haríamos con cualquier valor del diccionario. En lugar de devolver un solo valor, obtenemos una lista de ingredientes:

pizza.py

```
# Almacene información sobre una pizza que se está ordenando.
en pizza = {
    'corteza': 'gruesa',
    'coberturas': ['champiñones', 'queso extra'],
    }
```

```
    # Resume el pedido.                                         "
v print(f"Pediste una pizza de {pizza['crust']}-crust "con los
        siguientes ingredientes:")

w para aderezo en pizza['toppings']:
        imprimir("\t" + cobertura)
```

Comenzamos en u con un diccionario que contiene información sobre una pizza que se ha pedido. Una clave en el diccionario es 'crust', y el valor asociado es la cadena 'thick'. La siguiente clave, 'ingredientes', tiene una lista como valor que almacena todos los ingredientes solicitados. En v resumimos el pedido antes de construir la pizza. Cuando necesite dividir una línea larga en una llamada print() , elija un punto apropiado en el que dividir la línea que se está imprimiendo y finalice la línea con comillas. Aplique sangría a la línea siguiente, agregue una comilla de apertura y continúe la cadena. Python combinará automáticamente todas las cadenas que encuentre dentro de los paréntesis. Para imprimir los toppings, escribimos un bucle for w. Para acceder a la lista de ingredientes, usamos la tecla 'toppings' y Python toma la lista de ingredientes del diccionario.

El siguiente resultado resume la pizza que planeamos construir:

```
Pediste una pizza de masa gruesa con los siguientes ingredientes:
    champiñones
    extra queso
```

Puede anidar una lista dentro de un diccionario cada vez que desee asociar más de un valor con una sola clave en un diccionario. En el ejemplo anterior de los lenguajes de programación favoritos, si tuviéramos que almacenar las respuestas de cada persona en una lista, las personas podrían elegir más de un lenguaje favorito. Cuando recorremos el diccionario, el valor asociado con cada persona sería una lista de idiomas en lugar de un solo idioma. Dentro del bucle for del diccionario , usamos otro bucle for para recorrer la lista de idiomas asociados con cada persona:

favorito _idiomas.py

```
u idiomas_favoritos = {
        'jen': ['pitón', 'rubí'],
        'sara': ['c'],
        'eduardo': ['rubí', 'ir'],
        'phil': ['pitón', 'haskell'],
        }

v para nombre, idiomas en favoritos_idiomas.elementos():
        Los idiomas favoritos de print(f"\n{name.title()} son:")
w para idioma en idiomas:
                imprimir(f"\t{idioma.título()}")
```

Como puede ver en u, el valor asociado con cada nombre ahora es una lista. Note que algunas personas tienen un idioma favorito y otras tienen

múltiples favoritos. Cuando recorremos el diccionario en v, usamos el nombre de la variable languages para contener cada valor del diccionario, porque sabemos que cada valor será una lista. Dentro del bucle del diccionario principal, usamos otro bucle for w para recorrer la lista de idiomas favoritos de cada persona.

Ahora cada persona puede enumerar tantos idiomas favoritos como quiera:

```
Los idiomas favoritos de Jen son:
    Pitón
    Rubí

Los idiomas favoritos de Sarah son:
    C

Los idiomas favoritos de Phil son:
    Pitón
    Haskell

Los idiomas favoritos de Edward son:
    Rubí
    Vamos
```

Para refinar aún más este programa, puede incluir una sentencia if al comienzo del ciclo for del diccionario para ver si cada persona tiene más de un idioma favorito examinando el valor de len(idiomas). Si una persona tiene más de un favorito, el resultado sería el mismo. Si la persona solo tiene un idioma favorito, puede cambiar la redacción para reflejar eso. Por ejemplo, podría decir que el idioma favorito de Sarah es C.

No debe anidar listas y diccionarios demasiado profundamente. Si está anidando elementos mucho más profundamente que lo que ve en los ejemplos anteriores o está trabajando con el código de otra persona con niveles significativos de anidamiento, lo más probable es que exista una forma más sencilla de resolver el problema.

Un diccionario en un diccionario

Puede anidar un diccionario dentro de otro diccionario, pero su código puede complicarse rápidamente cuando lo hace. Por ejemplo, si tiene varios usuarios para un sitio web, cada uno con un nombre de usuario único, puede usar los nombres de usuario como claves en un diccionario. A continuación, puede almacenar información sobre cada usuario mediante el uso de un diccionario como el valor asociado con su nombre de usuario. En la siguiente lista, almacenamos tres datos sobre cada usuario: su nombre, apellido y ubicación. Accederemos a esta información recorriendo los nombres de usuario y el diccionario de información asociado con cada nombre de usuario:

muchos_usuarios.py

```
usuarios = {
    'aeinstein': {
        'primero': 'alberto',
```

```
            'último': 'einstein',
            'ubicación': 'princeton',
            },

        'mcurio': {
            'primero': 'marie',
            'último': 'curio',
            'ubicación': 'parís',
            },

        }

u para nombre de usuario, user_info en users.items():
v print(f"\nNombre de usuario: {nombre de usuario}")
w nombre_completo = f"{información_usuario['primero']} {información_usuario['último']}"
        ubicación = info_usuario['ubicación']

x print(f"\tNombre completo: {nombre_completo.título()}")
        print(f"\tUbicación: {ubicación.título()}")
```

Primero definimos un diccionario llamado usuarios con dos claves: una para los nombres de usuario 'aeinstein' y 'mcurie'. El valor asociado con cada clave es un diccionario que incluye el nombre, el apellido y la ubicación de cada usuario. en ti recorremos el diccionario de usuarios . Python asigna cada clave a la variable nombre de usuario, y el diccionario asociado con cada nombre de usuario se asigna a la variable user_info. Una vez dentro del bucle del diccionario principal, imprimimos el nombre de usuario en v.

En w comenzamos a acceder al diccionario interno. La variable user_info, que contiene el diccionario de información del usuario, tiene tres claves: 'primero', 'último' y 'ubicación'. Usamos cada clave para generar un nombre completo y una ubicación bien formateados para cada persona, y luego imprimimos un resumen de lo que sabemos sobre cada usuario x:

```
Nombre de usuario: einstein
    Nombre completo: Albert Einstein
    Ubicación: Princeton

Nombre de usuario: mcurie
    Nombre completo: Marie Curie
    Ubicación: París
```

Observe que la estructura del diccionario de cada usuario es idéntica. Aunque Python no lo requiere, esta estructura facilita el trabajo con los diccionarios anidados. Si el diccionario de cada usuario tuviera claves diferentes, el código dentro del ciclo for sería más complicado.

Inténtalo tú mismo

6-7. Gente: comience con el programa que escribió para el ejercicio 6-1 (página 99). Cree dos nuevos diccionarios que representen a diferentes personas y almacene los tres diccionarios en una lista llamada personas. Recorra su lista de personas. Mientras recorre la lista, imprima todo lo que sabe sobre cada persona.

6-8. Mascotas: haga varios diccionarios, donde cada diccionario represente una mascota diferente. En cada diccionario, incluya el tipo de animal y el nombre del dueño. Guarde estos diccionarios en una lista llamada mascotas. A continuación, recorra su lista y, mientras lo hace, imprima todo lo que sepa sobre cada mascota.

6-9. Lugares favoritos: haga un diccionario llamado lugares_favoritos. Piense en tres nombres para usar como claves en el diccionario y almacene de uno a tres lugares favoritos para cada persona. Para hacer este ejercicio un poco más interesante, pídales a algunos amigos que mencionen algunos de sus lugares favoritos. Recorre el diccionario e imprime el nombre de cada persona y sus lugares favoritos.

6-10. Números favoritos: Modifique su programa del Ejercicio 6-2 (página 99) para que cada persona pueda tener más de un número favorito. Luego escriba el nombre de cada persona junto con sus números favoritos.

6-11. Ciudades: Haz un diccionario llamado ciudades. Usa los nombres de tres ciudades como claves en tu diccionario. Cree un diccionario de información sobre cada ciudad e incluya el país en el que se encuentra la ciudad, su población aproximada y un dato sobre esa ciudad. Las claves para el diccionario de cada ciudad deben ser algo como país, población y hecho. Imprime el nombre de cada ciudad y toda la información que tengas guardada sobre ella.

6-12. Extensiones: ahora estamos trabajando con ejemplos que son lo suficientemente complejos como para que puedan extenderse de varias maneras. Utilice uno de los programas de ejemplo de este capítulo y amplíelo agregando nuevas claves y valores, cambiando el contexto del programa o mejorando el formato de la salida.

Resumen

En este capítulo aprendió cómo definir un diccionario y cómo trabajar con la información almacenada en un diccionario. Aprendió cómo acceder y modificar elementos individuales en un diccionario y cómo recorrer toda la información en un diccionario. Aprendió a recorrer los pares clave-valor de un diccionario, sus claves y sus valores. También aprendió cómo anidar varios diccionarios en una lista, anidar listas en un diccionario y anidar un diccionario dentro de un diccionario.

En el próximo capítulo aprenderá sobre los bucles while y cómo aceptar entradas de personas que están usando sus programas. Este será un capítulo emocionante, porque aprenderá a hacer que todos sus programas sean interactivos: podrán responder a las entradas del usuario.

7

Entrada de usuario y bucles mientras



La mayoría de los programas están escritos para resolver el problema de un usuario final. Para hacerlo, generalmente necesita obtener cierta información del usuario. Para un ejemplo simple, digamos que alguien quiere saber si tiene la edad suficiente para votar. Si escribe un programa para responder a esta pregunta, necesita saber la edad del usuario antes de poder dar una respuesta. El programa deberá pedirle al usuario que ingrese, o *ingrese,* su edad; una vez que el programa tiene esta entrada, puede compararla con la edad de votación para determinar si el usuario tiene la edad suficiente y luego informar el resultado.

En este capítulo, aprenderá cómo aceptar la entrada del usuario para que su programa pueda trabajar con ella. Cuando su programa necesite un nombre, podrá pedirle al usuario un nombre. Cuando su programa necesite una lista de nombres, podrá solicitar al usuario una serie de nombres. Para ello, utilizará la función input() .

También aprenderá cómo hacer que los programas sigan ejecutándose todo el tiempo que los usuarios quieran, para que puedan ingresar toda la información que necesiten; entonces, su programa puede trabajar con esa información. Utilizará el ciclo while de Python para mantener los programas en ejecución siempre que se cumplan ciertas condiciones.

Con la capacidad de trabajar con la entrada del usuario y la capacidad de controlar cuánto tiempo se ejecutan sus programas, podrá escribir programas totalmente interactivos.

Cómo funciona la función input()

La función input() pausa su programa y espera a que el usuario ingrese algún texto. Una vez que Python recibe la entrada del usuario, asigna esa entrada a una variable para que sea conveniente para usted trabajar con ella.

Por ejemplo, el siguiente programa le pide al usuario que ingrese algún texto, luego muestra ese mensaje al usuario:

loro.py

```
mensaje = entrada ("Dime algo y te lo repetiré:")
imprimir (mensaje)
```

La función input() toma un argumento: el *mensaje* o las instrucciones que queremos mostrar al usuario para que sepa qué hacer. En este ejemplo, cuando Python ejecuta la primera línea, el usuario ve el mensaje Dime algo y te lo repetiré: . El programa espera mientras el usuario ingresa su respuesta y continúa después de que el usuario presiona enter. La respuesta se asigna al mensaje variable, luego print (mensaje) muestra la entrada al usuario:

```
Dime algo y te lo repetiré: ¡Hola a todos!
¡Hola a todos!
```

Sublime Text y muchos otros editores no ejecutan programas que solicitan al usuario que ingrese información. Puede usar estos editores para escribir programas que soliciten una entrada, pero deberá ejecutar estos programas desde una terminal. Consulte “Ejecución de programas de Python desde una terminal” en la página 12.

Escribir indicaciones claras

Cada vez que utilice la función input() , debe incluir un aviso claro y fácil de seguir que le diga al usuario exactamente qué tipo de información está buscando. Cualquier declaración que le diga al usuario qué ingresar debería funcionar. Por ejemplo:

saludador.py

```
nombre = entrada ("Por favor ingrese su nombre:")
print(f"\nHola, {nombre}!")
```

Agregue un espacio al final de sus indicaciones (después de los dos puntos en el ejemplo anterior) para separar la indicación de la respuesta del usuario y dejarle claro a su usuario dónde ingresar su texto. Por ejemplo:

```
Por favor, introduzca su nombre: Eric
¡Hola, Eric!
```

A veces, querrá escribir un aviso que sea más largo que una línea. Por ejemplo, es posible que desee decirle al usuario por qué está solicitando cierta entrada. Puede asignar su aviso a una variable y pasar esa variable a la función input() . Esto le permite construir su solicitud en varias líneas, luego escribir una declaración de entrada () limpia .

saludador.py

```
prompt = "Si nos dice quién es, podemos personalizar los mensajes que ve".
prompt += "\n¿Cuál es tu nombre?

nombre = entrada (solicitud)
print(f"\nHola, {nombre}!")
```

Este ejemplo muestra una forma de construir una cadena de varias líneas. la primera línea asigna la primera parte del mensaje a la variable prompt. En la segunda línea, el operador += toma la cadena que se asignó a prompt y agrega la nueva cadena al final.

El mensaje ahora ocupa dos líneas, nuevamente con un espacio después del signo de interrogación para mayor claridad:

```
Si nos dices quién eres, podemos personalizar los mensajes que ves.
¿Cuál es tu primer nombre? eric

¡Hola, Eric!
```

Uso de int() para aceptar entradas numéricas

Cuando usa la función input() , Python interpreta todo lo que el usuario ingresa como una cadena. Considere la siguiente sesión de interpretación, que solicita la edad del usuario:

```
>>> edad = entrada ("¿Cuántos años tienes?")
¿Cuantos años tienes? 21
>>> edad
'21'
```

El usuario ingresa el número 21, pero cuando le preguntamos a Python por el valor de la edad, devuelve '21', la representación de cadena del valor numérico ingresado. Sabemos que Python interpretó la entrada como una cadena porque el número ahora está entre comillas. Si todo lo que quiere hacer es imprimir la entrada, esto funciona bien. Pero si intenta usar la entrada como un número, obtendrá un error:

```
>>> edad = entrada ("¿Cuántos años tienes?")
¿Cuantos años tienes? 21
tu >>> edad >= 18
Rastreo (llamadas recientes más última):
  Archivo "<stdin>", línea 1, en <módulo> v
TypeError: tipos no ordenados: str() >= int()
```

Cuando intenta usar la entrada para hacer una comparación numérica u, Python produce un error porque no puede comparar una cadena con un número entero: la cadena '21' asignada a la edad no se puede comparar con el valor numérico 18 v .

Podemos resolver este problema usando la función int() , que le dice a Python que trate la entrada como un valor numérico. La función int() convierte una representación de cadena de un número en una representación numérica, como se muestra aquí:

```
>>> edad = entrada ("¿Cuántos años tienes?")
¿Cuantos años tienes? 21
u >>> edad = int(edad)
>>> edad >= 18
Verdadero
```

En este ejemplo, cuando ingresamos 21 en el indicador, Python interpreta el número como una cadena, pero luego el valor se convierte en una representación numérica mediante int() u. Ahora Python puede ejecutar la prueba condicional: compara la edad (que ahora representa el valor numérico 21) y 18 para ver si la edad es mayor o igual a 18. Esta prueba se evalúa como Verdadero.

¿Cómo se usa la función int() en un programa real? Considere un programa que determina si las personas son lo suficientemente altas como para subirse a una montaña rusa:

```
montaña rusa.py altura = entrada ("¿Qué altura tienes, en pulgadas?")
altura = int(altura)

si altura >= 48:
    print("\nEres lo suficientemente alto para montar!")
demás:
    print("\nPodrás montar cuando seas un poco mayor.")
```

El programa puede comparar la altura con 48 porque altura = int(altura) convierte el valor de entrada en una representación numérica antes de realizar la comparación. Si el número ingresado es mayor o igual a 48, le decimos al usuario que es lo suficientemente alto:

```
¿Cuánto mides, en pulgadas? 71

¡Eres lo suficientemente alto para montar!
```

Cuando usa entradas numéricas para hacer cálculos y comparaciones, asegúrese de convertir primero el valor de entrada a una representación numérica.

El operador de módulo

Una herramienta útil para trabajar con información numérica es el *operador módulo* (%), que divide un número entre otro número y devuelve el resto:

```
>>> 4 % 3
1
```

```
>>> 5 % 3
2 >>> 6 % 3
0 >>> 7 % 3
1
```

El operador módulo no te dice cuántas veces cabe un número en otro; simplemente te dice cuál es el resto.

Cuando un número es divisible por otro número, el resto es 0, por lo que el operador módulo siempre devuelve 0. Puedes usar este hecho para determinar si un número es par o impar:

par_o_impar.py

```
número = entrada("Ingresa un número y te diré si es par o impar: ")
numero = int(numero)

si número % 2 == 0:
    print(f"\nEl número {número} es par.")
demás:
    print(f"\nEl número {número} es impar.")
```

Los números pares siempre son divisibles por dos, por lo que si el módulo de un número y dos es cero (aquí, si número % 2 == 0) el número es par. De lo contrario, es raro.

```
Introduce un número y te diré si es par o impar: 42

El número 42 es par.
```

Inténtalo tú mismo

7-1. Coche de alquiler: escriba un programa que le pregunte al usuario qué tipo de coche de alquiler le gustaría. Imprima un mensaje sobre ese automóvil, como "Déjame ver si puedo encontrarte un Subaru".

7-2. Asientos en el restaurante: escriba un programa que le pregunte al usuario cuántas personas hay en su grupo de cena. Si la respuesta es más de ocho, imprima un mensaje diciendo que tendrán que esperar por una mesa. De lo contrario, informe que su mesa está lista.

7-3. Múltiplos de diez: solicite al usuario un número y luego informe si el número es un múltiplo de 10 o no.

Introducción a los bucles while

El bucle for toma una colección de elementos y ejecuta un bloque de código una vez para cada elemento de la colección. Por el contrario, el bucle while se ejecuta *mientras* una determinada condición sea verdadera.

El ciclo while en acción

Puede usar un bucle while para contar una serie de números. Por ejemplo, el siguiente ciclo while cuenta de 1 a 5:

contando.py

```
número_actual = 1
mientras número_actual <= 5:
    imprimir (número_actual)
    número_actual += 1
```

En la primera línea, empezamos a contar desde 1 asignando número_actual el valor 1. El ciclo while se configura para seguir ejecutándose siempre que el valor de número_actual sea menor o igual a 5. El código dentro del ciclo imprime el valor de número_actual y luego agrega 1 a ese valor con número_actual += 1. (El operador += es una abreviatura de número_actual = número_actual + 1).

Python repite el bucle siempre que la condición número_actual <= 5 sea verdadera. Como 1 es menor que 5, Python imprime 1 y luego suma 1, haciendo el número actual 2. Como 2 es menor que 5, Python imprime 2 y vuelve a sumar 1, haciendo que el número actual sea 3, y así sucesivamente. Una vez que el valor de current_number es mayor que 5, el ciclo deja de ejecutarse y el programa finaliza:

```
1
2
3
4
5
```

Es muy probable que los programas que usa todos los días contengan bucles while . Por ejemplo, un juego necesita un bucle while para seguir ejecutándose todo el tiempo que desee seguir jugando y, por lo tanto, puede dejar de ejecutarse tan pronto como le pida que lo abandone. Los programas no serían divertidos de usar si dejaran de ejecutarse antes de que se lo indiquemos o continuaran ejecutándose incluso después de que quisiéramos salir, por lo que los bucles while son muy útiles.

Permitir que el usuario elija cuándo salir

Podemos hacer que el programa *parrot.py* se ejecute todo el tiempo que el usuario desee colocando la mayor parte del programa dentro de un ciclo while . Definiremos un *valor* de salida y luego mantendremos el programa ejecutándose mientras el usuario no haya ingresado el valor de salida:

loro.py

```
u prompt = "\nDime algo y te lo repetiré:"                "
prompt += "\nIngrese 'salir' para finalizar el programa.
```

```
                    ""
v mensaje =
w while mensaje != 'salir':
    mensaje = entrada (solicitud)
    imprimir (mensaje)
```

En u, definimos un aviso que le dice al usuario sus dos opciones: ingresar un mensaje o ingresar el valor de salida (en este caso, 'salir'). Luego configuramos un mensaje variable v para realizar un seguimiento de cualquier valor que ingrese el usuario. Definimos el mensaje como una cadena vacía, "", por lo que Python tiene algo que verificar la primera vez que llega a la línea while . La primera vez que se ejecuta el programa y Python llega a la declaración while , necesita comparar el valor del mensaje con 'quit', pero aún no se ha ingresado ninguna entrada del usuario. Si Python no tiene nada que comparar, no podrá continuar ejecutando el programa. Para resolver este problema, nos aseguramos de darle al mensaje un valor inicial. Aunque es solo una cadena vacía, tendrá sentido para Python y le permitirá realizar la comparación que hace que el ciclo while funcione. Este bucle while w se ejecuta siempre que el valor del mensaje no sea 'salir'.

La primera vez que pasa por el bucle, el mensaje es solo una cadena vacía, por lo que Python entra en el bucle. En mensaje = entrada (solicitud), Python muestra la solicitud y espera a que el usuario ingrese su entrada. Todo lo que ingresan se asigna a un mensaje y se imprime; luego, Python reevalúa la condición en el while
declaración. Siempre que el usuario no haya ingresado la palabra 'salir', el aviso se muestra nuevamente y Python espera más información. Cuando el usuario finalmente ingresa 'salir', Python deja de ejecutar el ciclo while y el programa finaliza:

Dime algo y te lo repetiré: Ingresa 'quit' para finalizar el
programa. **¡Hola a todos!**
¡Hola a todos!

Dime algo y te lo repetiré: Ingresa 'quit' para finalizar el
programa. **Hola de nuevo.**
Hola de nuevo.

Dime algo y te lo repetiré: Ingresa 'quit' para finalizar el
programa. **dejar**
dejar

Este programa funciona bien, excepto que imprime la palabra 'salir' como si fuera un mensaje real. Una simple prueba if soluciona esto:

```
prompt = "\nDime algo y te lo repetiré:"                    "
prompt += "\nIngrese 'salir' para finalizar el programa.
                    ""
mensaje
= while mensaje != 'salir':
    mensaje = entrada (solicitud)

    si el mensaje! = 'salir':
        imprimir (mensaje)
```

Ahora el programa realiza una comprobación rápida antes de mostrar el mensaje. y solo imprime el mensaje si no coincide con el valor de salida:

```
Dime algo y te lo repetiré: Ingresa 'quit' para finalizar el
programa. ¡Hola a todos!
¡Hola a todos!

Dime algo y te lo repetiré: Ingresa 'quit' para finalizar el
programa. Hola de nuevo.
Hola de nuevo.

Dime algo y te lo repetiré: Ingresa 'quit' para finalizar el
programa. dejar
```

usando una bandera

En el ejemplo anterior, hicimos que el programa realizara ciertas tareas mientras una condición determinada era verdadera. Pero, ¿qué pasa con los programas más complicados en los que muchos eventos diferentes pueden hacer que el programa deje de ejecutarse?

Por ejemplo, en un juego, varios eventos diferentes pueden terminar el juego. Cuando el jugador se queda sin barcos, se acaba el tiempo o las ciudades que se suponía que debían proteger son destruidas, el juego debería terminar. Debe terminar si ocurre cualquiera de estos eventos. Si pueden ocurrir muchos eventos posibles para detener el programa, trate de probar todas estas condiciones en un momento declaración se vuelve complicada y difícil.

Para un programa que debe ejecutarse solo mientras muchas condiciones sean verdaderas, puede definir una variable que determine si todo el programa está activo o no. Esta variable, llamada *bandera,* actúa como una señal para el programa. Podemos escribir nuestros programas para que se ejecuten mientras el indicador está establecido en Verdadero y dejar de ejecutarse cuando cualquiera de varios eventos establece el valor del indicador en Falso. Como resultado, nuestra declaración while general necesita verificar solo una condición: si la bandera es actualmente True o no. Luego, todas nuestras otras pruebas (para ver si ha ocurrido un evento que debería establecer el indicador en Falso) se pueden organizar ordenadamente en el resto del programa.

Agreguemos una bandera a *parrot.py* de la sección anterior. Este indicador, al que llamaremos activo (aunque puede llamarlo como quiera), controlará si el programa debe continuar ejecutándose o no:

```
prompt = "\nDime algo y te lo repetiré:"                    "
prompt += "\nIngrese 'salir' para finalizar el programa.

u activo = Verdadero
v mientras está activo:
        mensaje = entrada (solicitud)

w si el mensaje == 'salir':
            activo = Falso
x más:
            imprimir (mensaje)
```

Establecemos la variable active en True u para que el programa comience en un estado activo. Hacerlo simplifica la instrucción while porque no se realiza ninguna comparación en la instrucción while en sí; la lógica se cuida en otras partes del programa. Mientras la variable activa siga siendo True, el bucle seguirá ejecutándose v.

En la instrucción if dentro del ciclo while , verificamos el valor del mensaje una vez que el usuario ingresa su entrada. Si el usuario ingresa 'salir' w, activamos a False, y el bucle while se detiene. Si el usuario ingresa algo que no sea 'salir' x, imprimimos su entrada como un mensaje.

Este programa tiene el mismo resultado que el ejemplo anterior donde colocamos la prueba condicional directamente en la instrucción while . Pero ahora que tenemos un indicador para indicar si el programa general está en un estado activo, sería fácil agregar más pruebas (como declaraciones elif) para eventos que deberían causar que active se convierta en False. Esto es útil en programas complicados como juegos en los que puede haber muchos eventos que deberían hacer que el programa deje de ejecutarse. Cuando cualquiera de estos eventos hace que la bandera activa se vuelva Falso, el bucle principal del juego se cerrará, se puede mostrar un mensaje de *Game Over* y se le puede dar al jugador la opción de volver a jugar.

Usar pausa para salir de un bucle

Para salir de un ciclo while inmediatamente sin ejecutar ningún código restante en el ciclo, independientemente de los resultados de cualquier prueba condicional, use la instrucción break . La instrucción break dirige el flujo de su programa; puede usarlo para controlar qué líneas de código se ejecutan y cuáles no, de modo que el programa solo ejecute el código que desee, cuando lo desee.

Por ejemplo, considere un programa que pregunta al usuario sobre los lugares que ha visitado. visitado. Podemos detener el ciclo while en este programa llamando a break tan pronto como el usuario ingrese el valor 'quit' :

ciudades.py

```
prompt = "\nIngrese el nombre de una ciudad que haya visitado:"
indicador += "\n(Ingrese 'salir' cuando haya terminado.) "

u mientras es cierto:
    ciudad = entrada (mensaje)

    if ciudad == 'salir':
        romper
    demás:
        print(f"¡Me encantaría ir a {ciudad.título()}!")
```

Un ciclo que comienza con while True u se ejecutará para siempre a menos que llegue a una declaración de interrupción . El ciclo en este programa continúa pidiéndole al usuario que ingrese los nombres de las ciudades en las que ha estado hasta que ingresa 'salir'. Cuando ingresan 'quit', se ejecuta la instrucción break , lo que hace que Python salga del ciclo:

```
Ingrese el nombre de una ciudad que haya visitado: (Ingrese 'salir'
cuando haya terminado). Nueva York
¡Me encantaría ir a Nueva York!
```

```
Ingrese el nombre de una ciudad que haya visitado: (Ingrese
'salir' cuando haya terminado). San Francisco ¡Me encantaría ir a
San Francisco!

Ingrese el nombre de una ciudad que haya visitado: (Ingrese
'salir' cuando haya terminado). salir
```

Puede utilizar la sentencia *break en cualquiera de los bucles de Python. Por ejemplo, podrías usar break para salir de un* bucle *for que está trabajando a través de una lista o un diccionario.*

Uso de continuar en un bucle

En lugar de salir de un bucle por completo sin ejecutar el resto de su código, puede usar la instrucción continuar para volver al principio del bucle en función del resultado de una prueba condicional. Por ejemplo, considere un ciclo que cuenta del 1 al 10 pero imprime solo los números impares en ese rango:

contando.py

```
número_actual = 0
mientras número_actual < 10:
u numero_actual += 1
    si numero_actual % 2 == 0:
        Seguir

    imprimir (número_actual)
```

Primero establecemos current_number en 0. Debido a que es menor que 10, Python ingresa al ciclo while . Una vez dentro del bucle, incrementamos el conteo en 1 en u, por lo que el número_actual es 1. La declaración if luego verifica el módulo del número_actual y 2. Si el módulo es 0 (lo que significa que el número_actual es divisible por 2), la instrucción continuar dice Python para ignorar el resto del bucle y volver al principio. Si el número actual no es divisible por 2, se ejecuta el resto del bucle y Python imprime el número actual:

```
1
3
5
7
9
```

Evitar bucles infinitos

Cada bucle while necesita una forma de dejar de ejecutarse para que no continúe ejecutándose para siempre. Por ejemplo, este ciclo de conteo debería contar del 1 al 5:

contando.py

```
X = 1
mientras que x <= 5:
    imprimir (x)
    X + = 1
```

Pero si accidentalmente omite la línea x += 1 (como se muestra a continuación), el ciclo se ejecutará para siempre:

```
# ¡Este ciclo se ejecuta para siempre!
X = 1
mientras que x <= 5:
    imprimir (x)
```

Ahora el valor de x comenzará en 1 pero nunca cambiará. Como resultado, la prueba condicional x <= 5 siempre se evaluará como True y el bucle while se ejecutará para siempre, imprimiendo una serie de 1, como este:

```
1
1
1
1
--recorte--
```

Cada programador escribe accidentalmente un ciclo while infinito desde el tiempo al tiempo, especialmente cuando los bucles de un programa tienen condiciones de salida sutiles. Si su programa se atasca en un ciclo infinito, presione ctrl-C o simplemente cierre la ventana de terminal que muestra la salida de su programa.

Para evitar escribir bucles infinitos, pruebe cada bucle while y asegúrese de que el bucle se detenga cuando lo espere. Si desea que su programa finalice cuando el usuario ingrese un cierto valor de entrada, ejecute el programa e ingrese ese valor. Si el programa no finaliza, analice la forma en que su programa maneja el valor que debería causar la salida del bucle. Asegúrese de que al menos una parte del programa pueda hacer que la condición del ciclo sea Falsa o que llegue a una declaración de interrupción .

N ota *ublime Text y algunos otros editores tienen una ventana de salida incrustada. Esto puede dificultar la detención de un bucle infinito y es posible que deba cerrar el editor para finalizar el bucle. Intente hacer clic en el área de salida del editor antes de presionar ctrl-C, y debería poder cancelar un bucle infinito.*

Inténtalo tú mismo

7-4. Ingredientes para pizza: escriba un ciclo que solicite al usuario que ingrese una serie de ingredientes para pizza hasta que ingrese un valor de 'salir' . A medida que ingresen cada ingrediente, imprima un mensaje que diga que agregará ese ingrediente a su pizza.

7-5. Boletos de cine: un cine cobra diferentes precios de boletos según la edad de la persona. Si una persona es menor de 3 años, la entrada es gratuita; si son entre 3 y 12, el boleto cuesta $10; y si son mayores de 12 años, el boleto cuesta $15. Escriba un ciclo en el que pregunte a los usuarios su edad y luego dígales el costo de su boleto de cine.

(continuado)

7-6. Tres salidas: Escriba diferentes versiones del Ejercicio 7-4 o del Ejercicio 7-5 que hagan cada uno de los siguientes al menos una vez:

•Utilice una prueba condicional en la instrucción while para detener el ciclo.

•Utilice una variable activa para controlar cuánto tiempo se ejecuta el bucle.

•Utilice una declaración de interrupción para salir del ciclo cuando el usuario ingrese un valor de 'salir' .

7-7. Infinito: escriba un ciclo que nunca termine y ejecútelo. (Para finalizar el ciclo, presione ctrl-C o cierre la ventana que muestra la salida).

Usar un bucle while con listas y diccionarios

Hasta ahora, hemos trabajado con solo una parte de la información del usuario a la vez. Recibimos la entrada del usuario y luego imprimimos la entrada o una respuesta. La próxima vez que pasemos por el ciclo while , recibiremos otro valor de entrada y responderemos a eso. Pero para realizar un seguimiento de muchos usuarios y piezas de información, necesitaremos usar listas y diccionarios con nuestros bucles while .

Un bucle for es efectivo para recorrer una lista, pero no debe modificar una lista dentro de un bucle for porque Python tendrá problemas para realizar un seguimiento de los elementos de la lista. Para modificar una lista mientras trabaja en ella, use un bucle while . El uso de bucles while con listas y diccionarios le permite recopilar, almacenar y organizar muchas entradas para examinarlas e informar sobre ellas más adelante.

Mover elementos de una lista a otra

Considere una lista de usuarios recién registrados pero no verificados de un sitio web. Después de verificar estos usuarios, ¿cómo podemos moverlos a una lista separada de usuarios confirmados? Una forma sería usar un bucle while para extraer usuarios de la lista de usuarios no confirmados mientras los verificamos y luego los agregamos a una lista separada de usuarios confirmados. Así es como se vería ese código:

confirmado_usuarios.py

```
# Comience con los usuarios que deben verificarse, # y una lista
vacía para retener a los usuarios confirmados.
u usuarios_no confirmados = ['alicia', 'brian', 'candace']
usuarios_confirmados = []

# Verifique cada usuario hasta que no haya más usuarios sin confirmar.
# Mueva cada usuario verificado a la lista de usuarios confirmados.
v mientras usuarios_no_confirmados:
w usuario_actual = usuarios_no confirmados.pop()

    print(f"Verificando usuario: {current_user.title()}")
x usuarios_confirmados.append(usuario_actual)
```

```
# Mostrar todos los usuarios confirmados.
print("\nLos siguientes usuarios han sido confirmados:")
para usuario_confirmado en usuarios_confirmados:
        imprimir (usuario_confirmado.título())
```

Comenzamos con una lista de usuarios no confirmados en u (Alice, Brian y Candace) y una lista vacía para contener usuarios confirmados. El ciclo while en v se ejecuta siempre que la lista unconfirmed_users no esté vacía. Dentro de este ciclo, la función pop() en w elimina a los usuarios no verificados uno a la vez desde el final de unconfirmed_users. Aquí, porque Candace es la última en unconfirmed_users lista, su nombre será el primero en ser eliminado, asignado a usuario_actual y agregado a la lista de usuarios_confirmados en x. El siguiente es Brian, luego Alice.

Simulamos la confirmación de cada usuario imprimiendo un mensaje de verificación y luego agregándolos a la lista de usuarios confirmados. A medida que se reduce la lista de usuarios no confirmados, crece la lista de usuarios confirmados. Cuando la lista de usuarios no confirmados está vacía, el bucle se detiene y se imprime la lista de usuarios confirmados:

```
Verificando usuario: Candace
Verificando usuario: Brian
Verificando usuario: Alicia

Los siguientes usuarios han sido confirmados:
Candace
Brian
Alicia
```

Eliminación de todas las instancias de valores específicos de una lista

En el Capítulo 3 usamos remove() para eliminar un valor específico de una lista. La función remove() funcionó porque el valor que nos interesaba solo aparecía una vez en la lista. Pero, ¿qué sucede si desea eliminar todas las instancias de un valor de una lista?

Digamos que tiene una lista de mascotas con el valor 'gato' repetido varias veces. Para eliminar todas las instancias de ese valor, puede ejecutar un ciclo while hasta que 'gato' ya no esté en la lista, como se muestra aquí:

mascotas.py

```
mascotas = ['perro', 'gato', 'perro', 'pez dorado', 'gato', 'conejo', 'gato']
imprimir (mascotas)

mientras que 'gato' en mascotas:
        mascotas.remove('gato')

imprimir (mascotas)
```

Comenzamos con una lista que contiene varias instancias de 'gato'. Después de imprimir la lista, Python ingresa al ciclo while porque encuentra el valor 'gato' en la lista

al menos una vez. Una vez dentro del ciclo, Python elimina la primera instancia de 'gato', regresa a la línea while y luego vuelve a ingresar al ciclo cuando encuentra que 'gato' todavía está en la lista. Elimina cada instancia de 'gato' hasta que el valor ya no está en la lista, momento en el que Python sale del ciclo e imprime la lista nuevamente:

```
['perro', 'gato', 'perro', 'pez dorado', 'gato', 'conejo', 'gato'] ['perro', 'perro', 'pez
dorado', 'conejo']
```

Llenar un diccionario con entrada de usuario

Puede solicitar tanta entrada como necesite en cada paso a través de un tiempo círculo. Hagamos un programa de sondeo en el que cada paso por el bucle solicite el nombre y la respuesta del participante. Guardaremos los datos que recopilamos en un diccionario, porque queremos conectar cada respuesta con un usuario en particular:

montaña
_encuesta.py

```
respuestas = {}

# Establecer una bandera para indicar que el sondeo está activo.
sondeo_activo = Verdadero

mientras sondeo_activo:
        # Solicitar el nombre y la respuesta de la persona.
u nombre = input("\n¿Cuál es tu nombre?")
        respuesta = entrada ("¿Qué montaña te gustaría escalar algún día?")

        # Guarda la respuesta en el diccionario.
v respuestas[nombre] = respuesta

        # Averigüe si alguien más va a realizar la encuesta.
w repetir = entrada ("¿Le gustaría dejar que otra persona responda? (sí/no)")
        si repite == 'no':
                polling_active = Falso

# El sondeo está completo. Mostrar los resultados.
print("\n--- Resultados de la encuesta ---")
x para nombre, respuesta en answers.items():
        print(f"{nombre} quisiera escalar {respuesta}.")
```

El programa primero define un diccionario vacío (respuestas) y establece un indicador (polling_active) para indicar que el sondeo está activo. Siempre que polling_active sea True, Python ejecutará el código en el bucle while .

Dentro del bucle, se le solicita al usuario que ingrese su nombre y una montaña que le gustaría escalar. Esa información se almacena en el diccionario de respuestas v, y se le pregunta al usuario si desea o no mantener activa la encuesta w. Si ingresan sí, el programa ingresa nuevamente al ciclo while . Si ingresan no, el indicador polling_active se establece en False, el bucle while deja de ejecutarse y el bloque de código final en x muestra los resultados de la encuesta.

Si ejecuta este programa e ingresa respuestas de muestra, debería ver un resultado como este:

```
¿Cómo te llamas? eric
¿Qué montaña te gustaría escalar algún día? Denali
¿Te gustaría dejar que otra persona responda? (sí/no) sí

¿Cómo te llamas? lynn
¿Qué montaña te gustaría escalar algún día? Pulgar del diablo ¿Te gustaría
dejar que otra persona responda? (sí/no) no

--- Resultados de la
encuesta --- A Lynn le gustaría escalar Devil's Thumb.
A Eric le gustaría escalar el Denali.
```

Inténtalo tú mismo

7-8. Deli: haga una lista llamada sandwich_orders y llénela con los nombres de varios sándwiches. Luego haz una lista vacía llamada emparedados_terminados. Recorra la lista de pedidos de sándwiches e imprima un mensaje para cada pedido, como Hice su sándwich de atún. A medida que se prepara cada sándwich, muévalo a la lista de sándwiches terminados. Después de que se hayan hecho todos los sándwiches, imprima un mensaje que enumere cada sándwich que se preparó.

7-9. Sin Pastrami: Usando la lista sandwich_orders del Ejercicio 7-8, asegúrese de que el sándwich 'pastrami' aparezca en la lista al menos tres veces. Agregue código cerca del comienzo de su programa para imprimir un mensaje que diga que la charcutería se quedó sin pastrami, y luego use un ciclo while para eliminar todas las apariciones de 'pastrami' de sandwich_orders. Asegúrese de que ningún sándwich de pastrami termine en sándwiches terminados.

7-10. Vacaciones de ensueño: escriba un programa que haga encuestas a los usuarios sobre las vacaciones de sus sueños. Escribe un mensaje similar a Si pudieras visitar un lugar en el mundo, ¿a dónde irías? Incluya un bloque de código que imprima los resultados de la encuesta.

Resumen

En este capítulo, aprendió a usar input() para permitir que los usuarios proporcionen su propia información en sus programas. Aprendió a trabajar con entrada de texto y numérica y cómo usar bucles while para hacer que sus programas se ejecuten tanto tiempo como los usuarios lo deseen. Viste varias formas de controlar el flujo de un ciclo while configurando un indicador activo , usando la instrucción break y

utilizando la instrucción continuar . Aprendió cómo usar un ciclo while para mover elementos de una lista a otra y cómo eliminar todas las instancias de un valor de una lista. También aprendió cómo se pueden usar los bucles while con los diccionarios .

En el Capítulo 8 aprenderá acerca de *las funciones.* Las funciones le permiten romper sus programas en partes pequeñas, cada una de las cuales hace un trabajo específico. Puede llamar a una función tantas veces como desee y puede almacenar sus funciones en archivos separados. Mediante el uso de funciones, podrá escribir un código más eficiente que sea más fácil de solucionar y mantener y que se pueda reutilizar en muchos programas diferentes.

8

Funciones

En este capítulo, aprenderá a escribir *funciones,* que son bloques de código con nombre que están diseñados para realizar un trabajo específico. Cuando desea realizar una tarea particular que ha definido en una función, *llama* a la función responsable de la misma. Si necesita realizar esa tarea varias veces a lo largo de su programa, no necesita escribir todo el código para la misma tarea una y otra vez; simplemente llama a la función dedicada a manejar esa tarea, y la llamada le dice a Python que ejecute el código dentro de la función. Descubrirá que el uso de funciones hace que sus programas sean más fáciles de escribir, leer, probar y corregir.

En este capítulo también aprenderá formas de pasar información a las funciones. Aprenderá a escribir ciertas funciones cuyo trabajo principal es mostrar información y otras funciones diseñadas para procesar datos y devolver un valor o conjunto de valores. Finalmente, aprenderá a almacenar funciones en archivos separados llamados *módulos* para ayudar a organizar sus archivos de programa principales.

Definición de una función

Aquí hay una función simple llamada greeting_user() que imprime un saludo:

saludador.py
```
def saludo_usuario():
v """Mostrar un saludo simple."""
w imprime("¡Hola!")

x saludo_usuario()
```

Este ejemplo muestra la estructura más simple de una función. la línea en ti usa la palabra clave def para informar a Python que está definiendo una función. Esta es la *definición de la función,* que le dice a Python el nombre de la función y, si corresponde, qué tipo de información necesita la función para hacer su trabajo. Los paréntesis contienen esa información. En este caso, el nombre de la función es greeting_user() y no necesita información para hacer su trabajo, por lo que sus paréntesis están vacíos. (Aún así, los paréntesis son obligatorios). Finalmente, la definición termina en dos puntos.

Cualquier línea sangrada que siga a def greeting_user(): constituye el *cuerpo* de la función. El texto en v es un comentario llamado *docstring,* que describe lo que hace la función. Las cadenas de documentos están encerradas entre comillas triples, que Python busca cuando genera documentación para las funciones en sus programas.

La línea print("¡Hola!") ÿ es la única línea de código real en el cuerpo de esta función, entonces greeting_user() solo tiene un trabajo: print("¡Hola!").

Cuando quiera usar esta función, llámela. Una *llamada de función* le dice a Python que ejecute el código en la función. Para *llamar* a una función, escribe el nombre de la función, seguido de cualquier información necesaria entre paréntesis, como se muestra en x. Debido a que no se necesita información aquí, llamar a nuestra función es tan simple como ingresar greeting_user(). Como era de esperar, imprime ¡Hola!:

```
¡Hola!
```

Pasar información a una función

Modificada ligeramente, la función greeting_user() no solo puede decirle al usuario ¡Hola! sino también saludarlos por su nombre. Para que la función haga esto, ingrese el nombre de usuario entre paréntesis de la definición de la función en def greeting_user(). Al agregar el nombre de usuario aquí, permite que la función acepte cualquier valor de nombre de usuario que especifique. La función ahora espera que proporcione un valor para el nombre de usuario cada vez que la llame. Cuando llamas a greeting_user(), puedes pasarle un nombre, como 'jesse', entre paréntesis:

```
def saludo_usuario(nombre de usuario):
    """Mostrar un saludo simple."""
    print(f"Hola, {nombre de usuario.título()}!")

saludar_usuario('jesse')
```

Ingresar a greeting_user('jesse') llama a greeting_user() y le da a la función el información que necesita para ejecutar la llamada print() . La función acepta el nombre que le pasó y muestra el saludo para ese nombre:

```
¡Hola, Jesse!
```

Del mismo modo, al ingresar greeting_user('sarah') llama a greeting_user(), pasa 'sarah', e imprime ¡Hola, Sarah! Puede llamar a greeting_user() tantas veces como desee y pasarle cualquier nombre que desee para producir un resultado predecible cada vez.

Argumentos y Parámetros

En la función de saludo_usuario() anterior , definimos saludo_usuario() para requerir un valor para la variable nombre de usuario. Una vez que llamamos a la función y le dimos la información (el nombre de una persona), imprimió el saludo correcto.

El nombre de usuario variable en la definición de greeting_user() es un ejemplo de un *parámetro,* una información que la función necesita para hacer su trabajo. El valor 'jesse' en greeting_user('jesse') es un ejemplo de un *argumento.* Un argumento es una pieza de información que se pasa de una llamada de función a una función. Cuando llamamos a la función, colocamos el valor con el que queremos que trabaje la función entre paréntesis. En este caso, el argumento 'jesse' se pasó a la función greeting_user() y el valor se asignó al parámetro nombre de usuario.

Nota *gente a veces habla de argumentos y parámetros indistintamente. No se sorprenda si ve las variables en una definición de función referidas como argumentos o las variables en una llamada de función referidas como parámetros.*

Inténtalo tú mismo

8-1. Mensaje: Escriba una función llamada mostrar_mensaje() que imprima una oración que diga a todos lo que está aprendiendo en este capítulo. Llame a la función y asegúrese de que el mensaje se muestre correctamente.

8-2. Libro Favorito: Escriba una función llamada libro_favorito() que acepte un parámetro, título. La función debe imprimir un mensaje, como Uno de mis libros favoritos es Alicia en el país de las maravillas. Llame a la función, asegurándose de incluir el título de un libro como argumento en la llamada a la función.

pasar argumentos

Debido a que una definición de función puede tener múltiples parámetros, una llamada de función puede necesitar múltiples argumentos. Puede pasar argumentos a sus funciones de varias maneras. Puede usar *argumentos posicionales,* que deben estar en

el mismo orden en que se escribieron los parámetros; *argumentos de palabras clave,* donde cada argumento consta de un nombre de variable y un valor; y listas y diccionarios de valores. Veamos cada uno de estos a su vez.

Argumentos posicionales

Cuando llama a una función, Python debe hacer coincidir cada argumento en la llamada a la función con un parámetro en la definición de la función. La forma más sencilla de hacerlo se basa en el orden de los argumentos proporcionados. Los valores emparejados de esta manera se denominan *argumentos posicionales.*

Para ver cómo funciona esto, considere una función que muestre información sobre mascotas. La función nos dice qué tipo de animal es cada mascota y el nombre de la mascota, como se muestra aquí:

mascotas.py

```
u def describe_mascota(tipo_animal, nombre_mascota):
        """Mostrar información sobre una mascota."""
        print(f"\nTengo un {animal_type}").
        print(f"El nombre de mi {animal_type} es {pet_name.title()}").

v describe_mascota('hámster', 'harry')
```

La definición muestra que esta función necesita un tipo de animal y el nombre del animal u. Cuando llamamos a describe_pet(), necesitamos proporcionar un tipo de animal y un nombre, en ese orden. Por ejemplo, en la llamada de función, el argumento 'hamster' se asigna al parámetro animal_type y el argumento 'harry' se asigna al parámetro pet_name v. En el cuerpo de la función, estos dos parámetros se utilizan para mostrar información sobre la mascota. siendo descrito.

El resultado describe a un hámster llamado Harry:

```
tengo un hamster
Mi hámster se llama Harry.
```

Llamadas de función múltiple

Puede llamar a una función tantas veces como sea necesario. Describir una segunda mascota diferente requiere solo una llamada más a describe_pet():

```
def describe_mascota(tipo_animal, nombre_mascota):
        """Mostrar información sobre una mascota."""
        print(f"\nTengo un {animal_type}").
        print(f"El nombre de mi {animal_type} es {pet_name.title()}").

describe_mascota('hámster', 'harry')
describe_pet('perro', 'willie')
```

En esta segunda llamada de función, pasamos describe_mascota() los argumentos 'perro' y 'willie'. Al igual que con el conjunto anterior de argumentos que usamos, Python hace coincidir 'perro' con el parámetro tipo_animal y 'willie' con el parámetro nombre_mascota.

Como antes, la función hace su trabajo, pero esta vez imprime valores para un perro llamado Willie. Ahora tenemos un hámster llamado Harry y un perro llamado Willie:

```
tengo un hamster
Mi hámster se llama Harry.

Tengo un perro.
El nombre de mi perro es Willie.
```

Llamar a una función varias veces es una forma muy eficiente de trabajar. El código que describe una mascota se escribe una vez en la función. Luego, cada vez que desee describir una nueva mascota, llame a la función con la información de la nueva mascota. Incluso si el código para describir una mascota se expandiera a diez líneas, aún podría describir una nueva mascota en una sola línea llamando a la función nuevamente.

Puede usar tantos argumentos posicionales como necesite en sus funciones. Python trabaja a través de los argumentos que proporciona al llamar a la función y hace coincidir cada uno con el parámetro correspondiente en la definición de la función.

Importancia del orden en los argumentos posicionales

Puede obtener resultados inesperados si mezcla el orden de los argumentos en una llamada de función cuando usa argumentos posicionales:

```
def describe_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {animal_type}").
    print(f"El nombre de mi {animal_type} es {pet_name.title()}").

describe_mascota('harry', 'hamster')
```

En esta llamada de función enumeramos primero el nombre y segundo el tipo de animal. Debido a que el argumento 'harry' aparece primero esta vez, ese valor se asigna al parámetro animal_type. Asimismo, 'hamster' se asigna a pet_name. Ahora tenemos un “harry” llamado “Hamster”:

```
yo tengo un harry
Mi Harry se llama Hamster.
```

Si obtiene resultados divertidos como este, asegúrese de que el orden de los argumentos en su llamada a la función coincida con el orden de los parámetros en la definición de la función.

Argumentos de palabras clave

Un *argumento de palabra clave* es un par de nombre y valor que pasa a una función. Asocias directamente el nombre y el valor dentro del argumento, así que cuando pasas el argumento a la función, no hay confusión (no terminarás

con un Harry llamado Hamster). Los argumentos de palabras clave lo liberan de tener que preocuparse por ordenar correctamente sus argumentos en la llamada a la función y aclaran el rol de cada valor en la llamada a la función.

Reescribamos *pets.py* usando argumentos de palabras clave para llamar a describe_pet():

```
def describe_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {animal_type}").
    print(f"El nombre de mi {animal_type} es {pet_name.title()}").

describe_pet(animal_type='hamster', pet_name='harry')
```

La función describe_pet() no ha cambiado. Pero cuando llamamos a la función, le decimos explícitamente a Python con qué parámetro debe coincidir cada argumento. Cuando Python lee la llamada a la función, sabe asignar el argumento 'hamster' al parámetro animal_type y el argumento 'harry' a pet_name. El resultado muestra correctamente que tenemos un hámster llamado Harry.

El orden de los argumentos de las palabras clave no importa porque Python sabe dónde debe ir cada valor. Las siguientes dos llamadas de función son equivalentes:

```
describe_pet(animal_type='hamster', pet_name='harry')
describe_pet(pet_name='harry', animal_type='hamster')
```

Cuando utilice argumentos de palabras clave, asegúrese de utilizar los nombres exactos de los parámetros en la definición de la función.

Valores predeterminados

Al escribir una función, puede definir un *valor predeterminado* para cada parámetro. Si se proporciona un argumento para un parámetro en la llamada a la función, Python usa el valor del argumento. Si no, utiliza el valor predeterminado del parámetro. Entonces, cuando define un valor predeterminado para un parámetro, puede excluir el argumento correspondiente que normalmente escribiría en la llamada a la función. El uso de valores predeterminados puede simplificar sus llamadas a funciones y aclarar las formas en que se usan normalmente sus funciones.

Por ejemplo, si observa que la mayoría de las llamadas a describe_pet() se utilizan para describir perros, puede establecer el valor predeterminado de animal_type en 'perro'. Ahora cualquiera que llame a describe_pet() para un perro puede omitir esa información:

```
def describe_mascota(nombre_mascota, tipo_animal='perro'):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {animal_type}").
    print(f"El nombre de mi {animal_type} es {pet_name.title()}").

describe_pet(pet_name='willie')
```

Cambiamos la definición de describe_pet() para incluir un valor predeterminado, 'perro', para animal_type. Ahora, cuando se llama a la función sin animal_type especificado, Python sabe usar el valor 'perro' para este parámetro:

```
Tengo un perro.
El nombre de mi perro es Willie.
```

Tenga en cuenta que se tuvo que cambiar el orden de los parámetros en la definición de la función. Debido a que el valor predeterminado hace innecesario especificar un tipo de animal como argumento, el único argumento que queda en la llamada de función es el nombre de la mascota. Python todavía interpreta esto como un argumento posicional, por lo que si se llama a la función solo con el nombre de una mascota, ese argumento coincidirá con el primer parámetro enumerado en la definición de la función. Esta es la razón por la que el primer parámetro debe ser pet_name.

La forma más sencilla de usar esta función ahora es proporcionar solo la información de un perro. nombre en la llamada de función:

```
describe_mascota('willie')
```

Esta llamada de función tendría el mismo resultado que el ejemplo anterior. El único argumento proporcionado es 'willie', por lo que se compara con el primer parámetro de la definición, pet_name. Debido a que no se proporciona ningún argumento para animal_type, Python usa el valor predeterminado 'perro'.

Para describir un animal que no sea un perro, podría usar una llamada de función como esta:

```
describe_pet(pet_name='harry', animal_type='hamster')
```

Debido a que se proporciona un argumento explícito para animal_type , Python ignorar el valor predeterminado del parámetro.

Cuando utiliza valores predeterminados, cualquier parámetro con un valor predeterminado debe aparecer después de todos los parámetros que no tienen valores predeterminados. Esto le permite a Python continuar interpretando correctamente los argumentos posicionales.

Llamadas a funciones equivalentes

Debido a que los argumentos posicionales, los argumentos de palabras clave y los valores predeterminados se pueden usar juntos, a menudo tendrá varias formas equivalentes de llamar a una función. Considere la siguiente definición para describe_pet() con un valor predeterminado provisto:

```
def describe_mascota(nombre_mascota, tipo_animal='perro'):
```

Con esta definición, siempre se debe proporcionar un argumento para pet_name, y este valor se puede proporcionar usando la palabra clave posicional o

formato. Si el animal que se describe no es un perro, se debe incluir un argumento para animal_type en la llamada, y este argumento también se puede especificar usando el formato posicional o de palabra clave.

Todas las siguientes llamadas funcionarían para esta función:

```
# Un perro llamado Willie.
describe_mascota('willie')
describe_pet(pet_name='willie')

# Un hamster llamado Harry.
describe_mascota('harry', 'hamster')
describe_pet(pet_name='harry', animal_type='hamster')
describe_pet(animal_type='hamster', pet_name='harry')
```

Cada una de estas llamadas de función tendría el mismo resultado que los ejemplos anteriores.

Realmente no importa qué estilo de llamada utilice. Siempre que sus llamadas de función produzcan la salida que desea, simplemente use el estilo que le resulte más fácil de entender.

Evitar errores de argumento

Cuando comience a usar funciones, no se sorprenda si encuentra errores sobre argumentos no coincidentes. Los argumentos no coincidentes ocurren cuando proporciona menos o más argumentos de los que necesita una función para hacer su trabajo. Por ejemplo, esto es lo que sucede si tratamos de llamar a describe_pet() sin argumentos:

```
def describe_mascota(tipo_animal, nombre_mascota):
    """Mostrar información sobre una mascota."""
    print(f"\nTengo un {animal_type}").
    print(f"El nombre de mi {animal_type} es {pet_name.title()}").

describir_mascota()
```

Python reconoce que falta algo de información en la función llamada, y el rastreo nos dice que:

```
    Rastreo (última llamada más reciente): u
Archivo "pets.py", línea 6, en <módulo>
v describe_pet() w
TypeError: describe_pet() faltan 2 argumentos posicionales requeridos: 'animal_
    tipo' y 'nombre_mascota'
```

En u, el rastreo nos dice la ubicación del problema, lo que nos permite mire hacia atrás y vea que algo salió mal en nuestra llamada de función. en v la llamada a la función infractora se escribe para que la veamos. En w el rastreo

nos dice que a la llamada le faltan dos argumentos e informa los nombres de los argumentos que faltan. Si esta función estuviera en un archivo separado, probablemente podríamos reescribir la llamada correctamente sin tener que abrir ese archivo y leer el código de la función.

Python es útil porque lee el código de la función por nosotros y nos dice los nombres de los argumentos que necesitamos proporcionar. Esta es otra motivación para darle a sus variables y funciones nombres descriptivos. Si lo hace, los mensajes de error de Python serán más útiles para usted y cualquier otra persona que pueda usar su código.

Si proporciona demasiados argumentos, debería obtener un seguimiento similar que pueda ayudarlo a hacer coincidir correctamente su llamada de función con la definición de función.

Inténtalo tú mismo

8-3. Camiseta: Escribe una función llamada make_shirt() que acepte una talla y el texto de un mensaje que debería estar impreso en la camiseta. La función debe imprimir una oración que resuma el tamaño de la camisa y el mensaje impreso en ella.

Llame a la función una vez usando argumentos posicionales para hacer una camisa. Llame a la función por segunda vez usando argumentos de palabras clave.

8-4. Camisas grandes: modifique la función make_shirt() para que las camisas sean grandes por defecto con un mensaje que diga Me encanta Python. Haga una camisa grande y una camisa mediana con el mensaje predeterminado, y una camisa de cualquier tamaño con un mensaje diferente. mensaje.

8-5. Ciudades: Escribe una función llamada describe_city() que acepte el nombre de una ciudad y su país. La función debe imprimir una oración simple, como Reykjavik está en Islandia. Asigne al parámetro para el país un valor predeterminado.

Llame a su función para tres ciudades diferentes, al menos una de las cuales no se encuentra en el país predeterminado.

Valores devueltos

Una función no siempre tiene que mostrar su salida directamente. En su lugar, puede procesar algunos datos y luego devolver un valor o un conjunto de valores. El valor que devuelve la función se llama *valor de retorno.* La declaración de retorno toma un valor desde dentro de una función y lo envía de vuelta a la línea que llamó a la función.

Los valores devueltos le permiten mover gran parte del trabajo duro de su programa a funciones, lo que puede simplificar el cuerpo de su programa.

Devolver un valor simple

Veamos una función que toma un nombre y apellido, y devuelve un nombre completo con un formato limpio:

formateado
_nombre.py

```
u def get_formatted_name(nombre, apellido):
        """Retorna un nombre completo, con un formato ordenado."""
v nombre_completo = f"{nombre} {apellido}"
w devuelve nombre_completo.título()

x músico = get_formatted_name('jimi', 'hendrix')
   imprimir (músico)
```

La definición de get_formatted_name() toma como parámetros un primero y un último tu nombre La función combina estos dos nombres, agrega un espacio entre ellos y asigna el resultado a nombre_completo v. El valor de nombre_completo se convierte en mayúsculas y minúsculas y luego se devuelve a la línea de llamada en w.

Cuando llama a una función que devuelve un valor, debe proporcionar una variable a la que se le pueda asignar el valor devuelto. En este caso, el valor devuelto se asigna a la variable músico en x. El resultado muestra un nombre perfectamente formateado formado por las partes del nombre de una persona:

```
Jimi Hendrix
```

Esto puede parecer mucho trabajo para obtener un nombre bien formateado cuando Podríamos haber escrito simplemente:

```
imprimir("Jimi Hendrix")
```

Pero cuando considera trabajar con un programa grande que necesita almacenar muchos nombres y apellidos por separado, funciones como get_formatted_name() volverse muy útil. Usted almacena el nombre y el apellido por separado y luego llama a esta función cada vez que desea mostrar un nombre completo.

Hacer un argumento opcional

A veces tiene sentido hacer que un argumento sea opcional para que las personas que usan la función puedan optar por proporcionar información adicional solo si así lo desean. Puede usar valores predeterminados para hacer que un argumento sea opcional.

Por ejemplo, digamos que queremos expandir get_formatted_name() para manejar también los segundos nombres. Un primer intento de incluir segundos nombres podría verse así:

```
def get_formatted_name(nombre, segundo nombre, apellido):
    """Retorna un nombre completo, con un formato ordenado."""
    nombre_completo = f"{primer_nombre} {segundo_nombre} {apellido}"
    devolver nombre_completo.título()

músico = get_formatted_name('john', 'lee', 'hooker')
imprimir (músico)
```

Esta función funciona cuando se le asigna un nombre, un segundo nombre y un apellido. La función toma las tres partes de un nombre y luego construye una cadena a partir de ellas. La función agrega espacios cuando corresponde y convierte el nombre completo en mayúsculas y minúsculas:

John Lee Hooker

Pero los segundos nombres no siempre son necesarios, y esta función, tal como está escrita, no funcionaría si intentara llamarla solo con un nombre y un apellido. Para hacer que el segundo nombre sea opcional, podemos darle al argumento segundo_nombre un valor predeterminado vacío e ignorar el argumento a menos que el usuario proporcione un valor. Para hacer que get_formatted_name() funcione sin un segundo nombre, establecemos el valor predeterminado de middle_name en una cadena vacía y lo movemos al final de la lista de parámetros:

```
u def get_formatted_name(first_name, last_name, second_name=''):
        """Retorna un nombre completo, con un formato ordenado."""
v si segundo_nombre:
            nombre_completo = f"{primer_nombre} {segundo_nombre} {apellido}"
en otro:
            nombre_completo = f"{nombre} {apellido}"
        devolver nombre_completo.título()

    músico = get_formatted_name('jimi', 'hendrix')
    imprimir (músico)

x músico = get_formatted_name('john', 'hooker', 'lee')
    imprimir (músico)
```

En este ejemplo, el nombre se construye a partir de tres partes posibles. Debido a que siempre hay un nombre y un apellido, estos parámetros se enumeran primero en la definición de la función. El segundo nombre es opcional, por lo que aparece en último lugar en la definición y su valor predeterminado es una cadena vacía u.

En el cuerpo de la función, verificamos si se ha proporcionado un segundo nombre. Python interpreta las cadenas no vacías como True, por lo que si middle_name se evalúa como True si hay un argumento de segundo nombre en la llamada a la función. v. Si se proporciona un segundo nombre, el primer nombre, el segundo nombre y el apellido se combinan para formar un nombre completo. . Luego, este nombre se cambia a mayúsculas y minúsculas y se devuelve a la línea de llamada de función, donde se asigna a la variable músico y se imprime. Si no se proporciona un segundo nombre, la cadena vacía falla en la prueba if y el bloque else ejecuta w. El nombre completo se crea con solo un nombre y un apellido, y el nombre formateado se devuelve a la línea de llamada donde se asigna al músico y se imprime.

Llamar a esta función con un nombre y apellido es sencillo. Si estamos usando un segundo nombre, sin embargo, debemos asegurarnos de que el segundo nombre sea el último argumento pasado para que Python coincida correctamente con los argumentos posicionales x.

Esta versión modificada de nuestra función funciona para personas con solo una primera y apellido, y también funciona para personas que tienen un segundo nombre:

```
Jimi Hendrix
John Lee Hooker
```

Los valores opcionales permiten que las funciones manejen una amplia gama de casos de uso mientras permiten que las llamadas a funciones sean lo más simples posible.

Devolver un diccionario

Una función puede devolver cualquier tipo de valor que necesite, incluidas estructuras de datos más complicadas, como listas y diccionarios. Por ejemplo, la siguiente función toma partes de un nombre y devuelve un diccionario que representa a una persona:

persona.py

```
def build_person(nombre, apellido):
    """Retornar un diccionario de información sobre una persona."""
u persona = {'primero': nombre, 'apellido': apellido}
v devolver persona

músico = build_person('jimi', 'hendrix')
w imprimir (músico)
```

La función build_person() toma un nombre y apellido, y coloca estos valores en un diccionario en u. El valor de first_name se almacena con la clave 'first', y el valor de last_name se almacena con la clave 'last'. Todo el diccionario que representa a la persona se devuelve en v. El valor de retorno se imprime en w con las dos piezas originales de información textual ahora almacenadas en un diccionario:

```
{'primero': 'jimi', 'último': 'hendrix'}
```

Esta función toma información textual simple y la coloca en una estructura de datos más significativa que le permite trabajar con la información más allá de simplemente imprimirla. Las cadenas 'jimi' y 'hendrix' ahora están etiquetadas como nombre y apellido. Puede extender fácilmente esta función para aceptar valores opcionales como un segundo nombre, una edad, una ocupación o cualquier otra información que desee almacenar sobre una persona. Por ejemplo, el siguiente cambio también le permite almacenar la edad de una persona:

```
def build_person(nombre, apellido, edad=Ninguno):
    """Retornar un diccionario de información sobre una persona."""
    persona = {'primero': nombre, 'apellido': apellido}
    si la edad:
        persona['edad'] = edad
    persona de regreso

músico = build_person('jimi', 'hendrix', edad=27)
imprimir (músico)
```

Agregamos un nuevo parámetro opcional edad a la definición de la función y asignamos al parámetro el valor especial Ninguno, que se usa cuando una variable no tiene un valor específico asignado. Puede pensar en Ninguno como un valor de marcador de posición. En las pruebas condicionales, Ninguno se evalúa como Falso. Si la llamada a la función incluye un valor para la edad, ese valor se almacena en el diccionario. Esta función siempre almacena el nombre de una persona, pero también se puede modificar para almacenar cualquier otra información que desee sobre una persona.

Usar una función con un ciclo while

Puede usar funciones con todas las estructuras de Python que ha aprendido hasta ahora. Por ejemplo, usemos la función get_formatted_name() con un tiempo
bucle para saludar a los usuarios de manera más formal. Aquí hay un primer intento de saludar a las personas usando su nombre y apellido:

saludador.py

```
def get_formatted_name(nombre, apellido):
    """Retorna un nombre completo, con un formato ordenado."""
    nombre_completo = f"{nombre} {apellido}"
    devolver nombre_completo.título()

# ¡Este es un bucle infinito!
mientras que es cierto:
u print("\nPor favor dime tu nombre:")
    f_name = input("Nombre: ")
    l_name = input("Apellido: ")

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHola, {nombre_formateado}!")
```

Para este ejemplo, usamos una versión simple de get_formatted_name() que no involucra segundos nombres. El ciclo while le pide al usuario que ingrese su nombre, y le solicitamos su nombre y apellido por separado u.

Pero hay un problema con este bucle while : no hemos definido una condición de salida. ¿Dónde coloca una condición de salida cuando solicita una serie de entradas? Queremos que el usuario pueda salir lo más fácilmente posible, por lo que cada mensaje debe ofrecer una forma de salir. La instrucción break ofrece una forma sencilla de salir del bucle en cualquiera de las indicaciones:

```
def get_formatted_name(nombre, apellido):
    """Retorna un nombre completo, con un formato ordenado."""
    nombre_completo = f"{nombre} {apellido}"
    devolver nombre_completo.título()

mientras que es cierto:
    print("\nPor favor dime tu nombre:")
    print("(ingrese 'q' en cualquier momento para salir)")

    f_name = input("Nombre: ")
    si f_nombre == 'q':
        romper
```

```
    l_name = input("Apellido: ")
    si l_nombre == 'q':
        romper

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHola, {nombre_formateado}!")
```

Agregamos un mensaje que informa al usuario cómo salir y luego salimos del bucle si el usuario ingresa el valor de salida en cualquiera de las indicaciones. Ahora el programa seguirá saludando a las personas hasta que alguien ingrese 'q' para cualquier nombre:

```
Por favor, dígame su nombre:
(ingrese 'q' en cualquier momento para salir)
Nombre: eric
Apellido: Mathes

¡Hola, Eric Mathes!

Por favor, dígame su nombre:
(ingrese 'q' en cualquier momento para salir)
Nombre: q
```

Inténtalo tú mismo

8-6. Nombres de ciudades: escriba una función llamada city_country() que tome el nombre de una ciudad y su país. La función debería devolver una cadena con el formato siguiente:

```
"Santiago, Chile"
```

Llame a su función con al menos tres pares de ciudades y países e imprima los valores que se devuelven.

8-7. Álbum: escriba una función llamada make_album() que cree un diccionario que describa un álbum de música. La función debe tomar el nombre de un artista y el título de un álbum, y debe devolver un diccionario que contenga estos dos datos. Utilice la función para crear tres diccionarios que representen diferentes álbumes. Imprima cada valor devuelto para mostrar que los diccionarios están almacenando la información del álbum correctamente.

Use Ninguno para agregar un parámetro opcional a make_album() que le permita almacenar la cantidad de canciones en un álbum. Si la línea de llamada incluye un valor para el número de canciones, agregue ese valor al diccionario del álbum. Realice al menos una nueva llamada de función que incluya la cantidad de canciones en un álbum.

8-8. Álbumes de usuario: comience con su programa del ejercicio 8-7. escribe un rato bucle que permite a los usuarios ingresar el artista y el título de un álbum. Una vez que tenga esa información, llame a make_album() con la entrada del usuario e imprima el diccionario que se crea. Asegúrese de incluir un valor de salida en el ciclo while .

Pasar una lista

A menudo le resultará útil pasar una lista a una función, ya sea una lista de nombres, números u objetos más complejos, como diccionarios. Cuando pasa una lista a una función, la función obtiene acceso directo al contenido de la lista. Usemos funciones para que trabajar con listas sea más eficiente.

Digamos que tenemos una lista de usuarios y queremos imprimir un saludo para cada uno. El siguiente ejemplo envía una lista de nombres a una función llamada greeting_users(), que saluda a cada persona en la lista individualmente:

saludar_usuarios.py

```
def saludo_usuarios(nombres):
    """Imprime un saludo sencillo para cada usuario de la lista."""
    para nombre en nombres:
        msg = f"Hola, {nombre.título()}!"
        imprimir (mensaje)

u nombres de usuario = ['hannah', 'ty', 'margot']
saludar_usuarios(nombres de usuario)
```

Definimos greeting_users() para que espere una lista de nombres, que asigna a los nombres de los parámetros. La función recorre la lista que recibe e imprime un saludo para cada usuario. En u definimos una lista de usuarios y luego pasamos la lista de nombres de usuario a greeting_users() en nuestra llamada de función:

```
¡Hola Hannah!
¡Hola, Ty!
¡Hola, Margot!
```

Esta es la salida que queríamos. Cada usuario ve un saludo personalizado y puede llamar a la función en cualquier momento que desee saludar a un grupo específico de usuarios.

Modificar una lista en una función

Cuando pasa una lista a una función, la función puede modificar la lista. Cualquier cambio realizado en la lista dentro del cuerpo de la función es permanente, lo que le permite trabajar de manera eficiente incluso cuando se trata de grandes cantidades de datos.

Considere una empresa que crea modelos impresos en 3D de diseños que envían los usuarios. Los diseños que deben imprimirse se almacenan en una lista y, después de imprimirse, se mueven a una lista separada. El siguiente código hace esto sin usar funciones:

imprimiendo _modelos.py

```
# Comience con algunos diseños que deben imprimirse.
unprinted_designs = ['carcasa del teléfono', 'colgante de robot', 'dodecaedro']
modelos_completados = []

# Simule la impresión de cada diseño, hasta que no quede ninguno.
# Mueva cada diseño a complete_models después de la impresión.
mientras que unprinted_designs:
    diseño_actual = diseños_no_impresos.pop()
```

```
        print(f"Modelo de impresión: {diseño_actual}")
        modelos_completados.append(diseño_actual)

# Mostrar todos los modelos completados.
print("\nSe han impreso los siguientes modelos:")
para modelo_completado en modelos_completados:
        imprimir (modelo_completado)
```

Este programa comienza con una lista de diseños que deben imprimirse y una lista vacía llamada complete_models a la que se moverá cada diseño después de que se haya impreso. Mientras los diseños permanezcan en unprinted_designs, mientras loop simula la impresión de cada diseño eliminando un diseño del final de la lista, almacenándolo en current_design y mostrando un mensaje de que se está imprimiendo el diseño actual. Luego agrega el diseño a la lista de modelos completos. Cuando el bucle termina de ejecutarse, se muestra una lista de los diseños que se han impreso:

```
Modelo de impresión: dodecaedro
Modelo de impresión: robot colgante
Modelo de impresión: caja del teléfono

Se han impreso los siguientes modelos: caja del
teléfono con colgante de robot dodecaedro
```

Podemos reorganizar este código escribiendo dos funciones, cada una de las cuales hace un trabajo específico. La mayor parte del código no cambiará; simplemente lo estamos estructurando con más cuidado. La primera función se encargará de imprimir los diseños, y la segunda resumirá las impresiones que se han realizado:

```
u def print_models(unprinted_designs, complete_models):
        """
        Simule la impresión de cada diseño, hasta que no quede ninguno.
        Mueva cada diseño a complete_models después de la impresión.
        """
        mientras que unprinted_designs:
                diseño_actual = diseños_no_impresos.pop()
                print(f"Modelo de impresión: {diseño_actual}")
                modelos_completados.append(diseño_actual)

v def mostrar_modelos_completos(modelos_completos):
        """Mostrar todos los modelos que se imprimieron."""
        print("\nSe han impreso los siguientes modelos:")
        para modelo_completado en modelos_completados:
                imprimir (modelo_completado)

unprinted_designs = ['carcasa del teléfono', 'colgante de robot', 'dodecaedro']
modelos_completados = []

imprimir_modelos (diseños_no_impresos, modelos_completados)
mostrar_modelos_completados(modelos_completados)
```

En u definimos la función print_models() con dos parámetros: una lista de diseños que deben imprimirse y una lista de modelos terminados. Dadas estas dos listas, la función simula la impresión de cada diseño al vaciar la lista de diseños no impresos y llenar la lista de modelos completos. En v definimos la función show_completed_models() con un parámetro: la lista de modelos completados. Dada esta lista, show_completed_models() muestra el nombre de cada modelo que se imprimió.

Este programa tiene la misma salida que la versión sin funciones, pero el código está mucho más organizado. El código que hace la mayor parte del trabajo se ha movido a dos funciones separadas, lo que hace que la parte principal del programa sea más fácil de entender. Mire el cuerpo del programa para ver cuánto más fácil es entender lo que está haciendo este programa:

```
unprinted_designs = ['carcasa del teléfono', 'colgante de robot', 'dodecaedro']
modelos_completados = []

imprimir_modelos (diseños_no_impresos, modelos_completados)
mostrar_modelos_completados(modelos_completados)
```

Configuramos una lista de diseños sin imprimir y una lista vacía que contendrá los modelos completos. Luego, como ya hemos definido nuestras dos funciones, todo lo que tenemos que hacer es llamarlas y pasarles los argumentos correctos. Llamamos a print_models() y le pasamos las dos listas que necesita; como se esperaba, print_models() simula la impresión de los diseños. Luego llamamos a show_completed_models() y le pasamos la lista de modelos completados para que pueda informar los modelos que se han impreso. Los nombres de funciones descriptivos permiten que otros lean este código y lo entiendan, incluso sin comentarios.

Este programa es más fácil de ampliar y mantener que la versión sin funciones. Si necesitamos imprimir más diseños más adelante, simplemente podemos llamar a print_models() nuevamente. Si nos damos cuenta de que el código de impresión debe modificarse, podemos cambiar el código una vez y nuestros cambios se realizarán en todas partes donde se llame a la función. Esta técnica es más eficiente que tener que actualizar el código por separado en varios lugares del programa.

Este ejemplo también demuestra la idea de que cada función debe tener un trabajo específico. La primera función imprime cada diseño y la segunda muestra los modelos completos. Esto es más beneficioso que usar una función para hacer ambos trabajos. Si está escribiendo una función y nota que la función está realizando demasiadas tareas diferentes, intente dividir el código en dos funciones. Recuerde que siempre puede llamar a una función desde otra función, lo que puede ser útil al dividir una tarea compleja en una serie de pasos.

Evitar que una función modifique una lista

A veces querrá evitar que una función modifique una lista. Por ejemplo, suponga que comienza con una lista de diseños sin imprimir y escribe una función para moverlos a una lista de modelos completos, como en el ejemplo anterior. Puede decidir que, aunque haya impreso todos los diseños, desea conservar la lista original de diseños no impresos para sus registros.

Pero debido a que movió todos los nombres de diseño fuera de unprinted_designs, la lista ahora está vacía y la lista vacía es la única versión que tiene; el original se ha ido. En este caso, puede solucionar este problema pasando a la función una copia de la lista, no el original. Cualquier cambio que la función haga en la lista afectará solo a la copia, dejando intacta la lista original.

Puede enviar una copia de una lista a una función como esta:

```
nombre_función(nombre_lista[:])
```

La notación de división [:] hace una copia de la lista para enviarla a la función. Si no quisiéramos vaciar la lista de diseños no impresos en print_models.py , podríamos llamar a print_models() así:

```
imprimir_modelos(diseños_no_impresos[:], modelos_completados)
```

La función print_models() puede hacer su trabajo porque todavía recibe el nombres de todos los diseños no impresos. Pero esta vez usa una copia de la lista original de diseños no impresos, no la lista actual de diseños no impresos . La lista complete_models se llenará con los nombres de los modelos impresos como lo hacía antes, pero la lista original de diseños no impresos no se verá afectada por la función.

Aunque puede conservar el contenido de una lista pasando una copia a sus funciones, debe pasar la lista original a funciones a menos que tenga una razón específica para pasar una copia. Es más eficiente que una función trabaje con una lista existente para evitar usar el tiempo y la memoria necesarios para hacer una copia separada, especialmente cuando trabaja con listas grandes.

Inténtalo tú mismo

8-9. Mensajes: haga una lista que contenga una serie de mensajes de texto cortos. Pase la lista a una función llamada show_messages(), que imprime cada mensaje de texto.

8-10. Envío de mensajes: comience con una copia de su programa del ejercicio 8-9. Escriba una función llamada enviar_mensajes() que imprima cada mensaje de texto y mueva cada mensaje a una nueva lista llamada mensajes_enviados a medida que se imprime. Después de llamar a la función, imprima ambas listas para asegurarse de que los mensajes se movieron correctamente.

8-11. Mensajes archivados: comience con su trabajo del ejercicio 8-10. Llame a la función send_messages() con una copia de la lista de mensajes. Después de llamar a la función, imprima ambas listas para mostrar que la lista original ha conservado sus mensajes.

Pasar un número arbitrario de argumentos

A veces, no sabrá de antemano cuántos argumentos debe aceptar una función. Afortunadamente, Python permite que una función recopile un número arbitrario de argumentos de la declaración de llamada.

Por ejemplo, considere una función que construye una pizza. Necesita aceptar una cantidad de ingredientes, pero no se puede saber de antemano cuántos ingredientes querrá una persona. La función del siguiente ejemplo tiene un parámetro, *toppings, pero este parámetro recopila tantos argumentos como proporciona la línea de llamada:

pizza.py

```
def hacer_pizza(*ingredientes):
    """Imprimir la lista de toppings que se han solicitado."""
    imprimir (coberturas)

hacer_pizza('pepperoni')
make_pizza('champiñones', 'pimientos verdes', 'queso extra')
```

El asterisco en el nombre del parámetro *toppings le dice a Python que haga una vaciar la tupla llamada toppings y empaquetar cualquier valor que reciba en esta tupla. La llamada a print() en el cuerpo de la función produce una salida que muestra que Python puede manejar una llamada de función con un valor y una llamada con tres valores. Trata las diferentes llamadas de manera similar. Tenga en cuenta que Python empaqueta los argumentos en una tupla, incluso si la función recibe solo un valor:

```
('pepperoni',)
('champiñones', 'pimientos verdes', 'queso extra')
```

Ahora podemos reemplazar la llamada print() con un bucle que se ejecuta a través del lista de ingredientes y describe la pizza que se pide:

```
def hacer_pizza(*ingredientes):
    """Resume la pizza que estamos a punto de hacer."""
    print("\nPreparando una pizza con los siguientes ingredientes:")
    para cubrir en coberturas:
        imprimir(f"- {topping}")

hacer_pizza('pepperoni')
make_pizza('champiñones', 'pimientos verdes', 'queso extra')
```

La función responde apropiadamente, ya sea que reciba un valor o tres valores:

```
Hacer una pizza con los siguientes ingredientes: - pepperoni


Elaboración de una pizza con los siguientes ingredientes: -
champiñones
- pimientos verdes
- queso extra
```

Esta sintaxis funciona sin importar cuántos argumentos reciba la función.

Mezclar argumentos posicionales y arbitrarios

Si desea que una función acepte varios tipos diferentes de argumentos, el parámetro que acepta un número arbitrario de argumentos debe colocarse en último lugar en la definición de la función. Python primero hace coincidir los argumentos posicionales y de palabras clave y luego recopila los argumentos restantes en el parámetro final.

Por ejemplo, si la función necesita tomar un tamaño para la pizza, ese El parámetro debe ir antes del parámetro *toppings:

```
def make_pizza(size, *toppings):
    """Resumir la pizza que estamos a punto de hacer."""
    print(f"\nHacer una pizza de {tamaño} pulgadas con los siguientes ingredientes:") para
    agregar ingredientes: print(f"- {topping}")


make_pizza(16, 'pepperoni')
make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

En la definición de la función, Python asigna el primer valor que recibe al tamaño del parámetro. Todos los demás valores que vienen después se almacenan en las coberturas de tupla. Las llamadas de función incluyen un argumento para el tamaño primero, seguido de tantos ingredientes como sea necesario.

Ahora cada pizza tiene un tamaño y una cantidad de ingredientes, y cada información se imprime en el lugar adecuado, mostrando el tamaño primero y los ingredientes después:

```
Hacer una pizza de 16 pulgadas con los siguientes ingredientes:
- pepperoni

Hacer una pizza de 12 pulgadas con los siguientes ingredientes:
- champiñones
- pimientos verdes
- queso extra
```

*A menudo verá el nombre de parámetro genérico *args, que recopila información posicional arbitraria . argumentos como este.*

Uso de argumentos de palabras clave arbitrarias

A veces querrá aceptar un número arbitrario de argumentos, pero no sabrá de antemano qué tipo de información se pasará a la función. En este caso, puede escribir funciones que acepten tantos pares clave-valor como proporcione la declaración de llamada. Un ejemplo implica la creación de perfiles de usuario: sabe que obtendrá información sobre un usuario, pero no está seguro de qué tipo de información recibirá. La función build_profile() en el

El siguiente ejemplo siempre incluye un nombre y un apellido, pero también acepta un número arbitrario de argumentos de palabras clave:

perfil_usuario.py

```
def build_profile(primero, último, **información_usuario):
    """Crear un diccionario que contenga todo lo que sabemos sobre un usuario."""
u user_info['first_name'] = primero
    user_info['last_name'] = último
    devolver información_de_usuario

perfil_usuario = build_profile('alberto', 'einstein',
                               ubicación='princeton',
                               campo='física')
imprimir (perfil_usuario)
```

La definición de build_profile() espera un nombre y apellido, y luego le permite al usuario pasar tantos pares de nombre-valor como quiera. Los asteriscos dobles antes del parámetro **user_info hacen que Python cree un diccionario vacío llamado user_info y empaquete cualquier par de nombre-valor que reciba en este diccionario. Dentro de la función, puede acceder a los pares de valores clave en user_info tal como lo haría con cualquier diccionario.

En el cuerpo de build_profile(), agregamos el nombre y apellido al diccionario user_info porque siempre recibiremos estas dos piezas de información del usuario u, y aún no se han colocado en el diccionario. Luego devolvemos el diccionario user_info a la línea de llamada de función.

Llamamos a build_profile(), pasándole el nombre de pila 'albert', el apellido 'einstein' y los dos pares clave-valor location='princeton' y field='physics'. Asignamos el perfil devuelto a user_profile e imprimimos user_profile:

```
{'ubicación': 'princeton', 'campo': 'física',
 'nombre': 'alberto', 'apellido': 'einstein'}
```

El diccionario devuelto contiene el nombre y apellido del usuario y, en este caso, la ubicación y el campo de estudio también. La función funcionaría sin importar cuántos pares clave-valor adicionales se proporcionen en la llamada a la función.

Puede mezclar valores posicionales, de palabra clave y arbitrarios de muchas maneras diferentes al escribir sus propias funciones. Es útil saber que existen todos estos tipos de argumentos porque los verá con frecuencia cuando comience a leer el código de otras personas. Se necesita práctica para aprender a usar los diferentes tipos correctamente y saber cuándo usar cada tipo. Por ahora, recuerde usar el enfoque más simple que haga el trabajo. A medida que progrese, aprenderá a usar el enfoque más eficiente cada vez.

*A menudo verá el nombre del parámetro **kwargs utilizado para recopilar argumentos de palabras clave no específicos.*

Inténtalo tú mismo

8-12. Sándwiches: escriba una función que acepte una lista de artículos que una persona quiere en un sándwich. La función debe tener un parámetro que recopile tantos elementos como proporcione la llamada a la función, y debe imprimir un resumen del sándwich que se está ordenando. Llame a la función tres veces, usando un número diferente de argumentos cada vez.

8-13. Perfil de usuario: Comience con una copia de user_profile.py de la página 149. Cree un perfil de usted mismo llamando a build_profile(), usando su nombre y apellido y otros tres pares clave-valor que lo describan.

8-14. Coches: escriba una función que almacene información sobre un coche en un diccionario.

La función siempre debe recibir un fabricante y un nombre de modelo. Entonces debería aceptar un número arbitrario de argumentos de palabras clave. Llame a la función con la información requerida y otros dos pares de nombre y valor, como un color o una característica opcional. Su función debería funcionar para una llamada como esta:

```
coche = make_car('subaru', 'outback', color='blue', tow_package=True)
```

Imprima el diccionario que se devuelve para asegurarse de que toda la información se almacenó correctamente.

Almacenamiento de sus funciones en módulos

Una ventaja de las funciones es la forma en que separan los bloques de código de su programa principal. Al usar nombres descriptivos para sus funciones, su programa principal será mucho más fácil de seguir. Puede ir un paso más allá almacenando sus funciones en un archivo separado llamado *módulo* y luego *importando* ese módulo en su programa principal. Una declaración de importación le dice a Python que haga que el código en un módulo esté disponible en el archivo de programa que se está ejecutando actualmente.

Almacenar sus funciones en un archivo separado le permite ocultar los detalles del código de su programa y enfocarse en su lógica de nivel superior. También le permite reutilizar funciones en muchos programas diferentes. Cuando almacena sus funciones en archivos separados, puede compartir esos archivos con otros programadores sin tener que compartir todo su programa. Saber cómo importar funciones también le permite usar bibliotecas de funciones que otros programadores han escrito.

Hay varias formas de importar un módulo y le mostraré cada una de ellas brevemente.

Importación de un módulo completo

Para comenzar a importar funciones, primero debemos crear un módulo. un *modulo* es un archivo que termina en .py que contiene el código que desea importar a su

programa. Hagamos un módulo que contenga la función make_pizza(). Para hacer este módulo, eliminaremos todo del archivo *pizza.py* excepto la función make_pizza():

pizza.py

```
def make_pizza(tamaño, *ingredientes):
    """Resume la pizza que estamos a punto de hacer."""
    print(f"\nPreparando una pizza de {tamaño} pulgadas con los siguientes ingredientes:")
    para cubrir en coberturas:
        imprimir(f"- {topping}")
```

Ahora crearemos un archivo separado llamado *making_pizzas.py* en el mismo directorio que *pizza.py.* Este archivo importa el módulo que acabamos de crear y luego realiza dos llamadas a make_pizza():

haciendo _pizzas.py

```
importar pizza

u pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

Cuando Python lee este archivo, la línea import pizza le dice a Python que abra el archivo *pizza.py* y copie todas las funciones de este en este programa. En realidad, no ve el código que se copia entre archivos porque Python copia el código detrás de escena justo antes de que se ejecute el programa. Todo lo que necesita saber es que cualquier función definida en *pizza.py* ahora estará disponible en *making_pizzas.py.*

Para llamar a una función desde un módulo importado, ingresa el nombre del módulo que importaste, pizza, seguido del nombre de la función, make_pizza(), separados por un punto u. Este código produce el mismo resultado que el programa original que no importó un módulo:

```
Hacer una pizza de 16 pulgadas con los siguientes ingredientes:
- pepperoni

Hacer una pizza de 12 pulgadas con los siguientes ingredientes:
- champiñones
- pimientos verdes
- queso extra
```

Este primer enfoque de importación, en el que simplemente escribe import seguido del nombre del módulo, hace que todas las funciones del módulo estén disponibles en su programa. Si usa este tipo de declaración de importación para importar un módulo completo llamado *module_name.py,* cada función en el módulo está disponible a través de la siguiente sintaxis:

nombre_módulo.nombre_función()

Importación de funciones específicas

También puede importar una función específica de un módulo. Aquí está la sintaxis general para este enfoque:

```
from nombre_módulo import nombre_función
```

Puede importar tantas funciones como desee de un módulo separando el nombre de cada función con una coma:

```
from nombre_módulo importar función_0 , función_1 , función_2
```

El ejemplo de *making_pizzas.py* se vería así si queremos importar solo la función que vamos a usar:

```
de pizza importar hacer_pizza

hacer_pizza(16, 'pepperoni')
make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

Con esta sintaxis, no necesita usar la notación de puntos cuando llama a una función. Debido a que importamos explícitamente la función make_pizza() en la declaración de importación , podemos llamarla por su nombre cuando usamos la función.

Uso de as para dar un alias a una función

Si el nombre de una función que está importando puede entrar en conflicto con un nombre existente en su programa o si el nombre de la función es largo, puede usar un *alias corto y único, un* nombre alternativo similar a un apodo para la función. Le dará a la función este apodo especial cuando importe la función.

Aquí le damos a la función make_pizza() un alias, mp(), importando hacer_pizza como mp. La palabra clave as cambia el nombre de una función utilizando el alias que proporciona:

```
de pizza importar make_pizza como mp

mp (16, 'pepperoni')
mp(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

La declaración de importación que se muestra aquí cambia el nombre de la función make_pizza() a mp() en este programa. Cada vez que queramos llamar a make_pizza() , simplemente podemos escribir mp() en su lugar, y Python ejecutará el código en make_pizza() mientras evita cualquier confusión con otra función make_pizza() que podría haber escrito en este archivo de programa.

La sintaxis general para proporcionar un alias es:

```
from nombre_módulo importar nombre_función como fn
```

Uso de as para dar un alias a un módulo

También puede proporcionar un alias para un nombre de módulo. Darle a un módulo un alias corto, como p para pizza, le permite llamar a las funciones del módulo más rápidamente. Llamar a p.make_pizza() es más conciso que llamar a pizza.make_pizza():

```
importar pizza como p

p.hacer_pizza(16, 'pepperoni')
p.make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

El módulo pizza recibe el alias p en la declaración de importación , pero todas las funciones del módulo conservan sus nombres originales. Llamar a las funciones escribiendo p.make_pizza() no solo es más conciso que escribir pizza.make_pizza(), sino que también redirige su atención del nombre del módulo y le permite concentrarse en los nombres descriptivos de sus funciones. Estos nombres de funciones, que le indican claramente lo que hace cada función, son más importantes para la legibilidad de su código que usar el nombre completo del módulo.

La sintaxis general para este enfoque es:

```
importar module_name como mn
```

Importación de todas las funciones en un módulo

Puede decirle a Python que importe cada función en un módulo usando el operador asterisco (*) :

```
de la importación de pizzas *

hacer_pizza(16, 'pepperoni')
make_pizza(12, 'champiñones', 'pimientos verdes', 'queso extra')
```

El asterisco en la declaración de importación le dice a Python que copie todas las funciones del módulo pizza en este archivo de programa. Debido a que cada función se importa, puede llamar a cada función por su nombre sin usar la notación de puntos. Sin embargo, es mejor no usar este enfoque cuando trabaja con módulos más grandes que no escribió: si el módulo tiene un nombre de función que coincide con un nombre existente en su proyecto, puede obtener algunos resultados inesperados. Python puede ver varias funciones o variables con el mismo nombre y, en lugar de importar todas las funciones por separado, las sobrescribirá.

El mejor enfoque es importar la función o funciones que desee, o importar el módulo completo y usar la notación de puntos. Esto conduce a un código claro que es fácil de leer y comprender. Incluyo esta sección para que reconozca declaraciones de importación como las siguientes cuando las vea en el código de otras personas:

```
de la importación de module_name *
```

Funciones de estilo

Debe tener en cuenta algunos detalles cuando esté diseñando funciones. Las funciones deben tener nombres descriptivos, y estos nombres deben usar letras minúsculas y guiones bajos. Los nombres descriptivos lo ayudan a usted y a otros a comprender lo que su código está tratando de hacer. Los nombres de los módulos también deben usar estas convenciones.

Cada función debe tener un comentario que explique de manera concisa lo que hace la función. Este comentario debería aparecer inmediatamente después de la definición de la función y usar el formato docstring. En una función bien documentada, otros programadores pueden usar la función leyendo solo la descripción en la cadena de documentación. Deberían poder confiar en que el código funciona como se describe y, siempre que sepan el nombre de la función, los argumentos que necesita y el tipo de valor que devuelve, deberían poder usarlo en sus programas.

Si especifica un valor predeterminado para un parámetro, no se deben usar espacios a cada lado del signo igual:

```
def nombre_función (parámetro_0, parámetro_1='valor predeterminado ')
```

Se debe usar la misma convención para los argumentos de palabras clave en las llamadas a funciones:

```
nombre_función(valor_0, parámetro_1='valor')
```

PEP 8 *(https:// www.python.org/ dev/ peps/ pep-0008/)* recomienda que limite las líneas de código a 79 caracteres para que cada línea sea visible en una ventana de editor de tamaño razonable. Si un conjunto de parámetros hace que la definición de una función tenga más de 79 caracteres, presione Intro después del paréntesis de apertura en la línea de definición. En la siguiente línea, presione tabulador dos veces para separar la lista de argumentos del cuerpo de la función, que solo tendrá una sangría de un nivel.

La mayoría de los editores alinean automáticamente cualquier línea adicional de parámetros para coincida con la sangría que ha establecido en la primera línea:

```
def nombre_función (
        parámetro_0, parámetro_1, parámetro_2,
        parámetro_3, parámetro_4, parámetro_5):
    cuerpo funcional...
```

Si su programa o módulo tiene más de una función, puede separar califique cada una con dos líneas en blanco para que sea más fácil ver dónde termina una función y comienza la siguiente.

Todas las declaraciones de importación deben escribirse al principio de un archivo. La única excepción es si usa comentarios al comienzo de su archivo para describir el programa general.

Inténtalo tú mismo

8-15. Modelos de impresión: coloque las funciones para el ejemplo modelos_impresión.py en un archivo separado llamado funciones_impresión.py. Escriba una declaración de importación en la parte superior de printing_models.py y modifique el archivo para usar las funciones importadas.

8-16. Importaciones: utilizando un programa que escribió que tiene una función, almacene esa función en un archivo separado. Importe la función a su archivo de programa principal y llame a la función utilizando cada uno de estos enfoques:

importar *module_name*
from *nombre_módulo* import *nombre_función*
from *nombre_módulo* importar *nombre_función* como *fn*
importar *module_name* como *mn*
de la importación de *module_name* *

8-17. Funciones de diseño: elija tres programas cualquiera que haya escrito para este capítulo y asegúrese de que sigan las pautas de diseño descritas en esta sección.

Resumen

En este capítulo aprendiste a escribir funciones y pasar argumentos para que tus funciones tengan acceso a la información que necesitan para hacer su trabajo. Aprendió a usar argumentos posicionales y de palabras clave, y a aceptar un número arbitrario de argumentos. Viste funciones que muestran resultados y funciones que devuelven valores. Aprendió a usar funciones con listas, diccionarios, declaraciones if y bucles while . También vio cómo almacenar sus funciones en archivos separados llamados *módulos,* por lo que sus archivos de programa serán más simples y fáciles de entender. Finalmente, aprendió a diseñar sus funciones para que sus programas continúen bien estructurados y sean tan fáciles de leer como sea posible para usted y otros.

Uno de sus objetivos como programador debe ser escribir código simple que haga lo que usted quiere, y las funciones lo ayudarán a hacerlo. Le permiten escribir bloques de código y dejarlos solos una vez que sabe que funcionan. Cuando sabe que una función hace su trabajo correctamente, puede confiar en que seguirá funcionando y pasará a su siguiente tarea de codificación.

Las funciones le permiten escribir código una vez y luego reutilizar ese código tantas veces como desee. Cuando necesite ejecutar el código en una función, todo lo que necesita hacer es escribir una llamada de una línea y la función hace su trabajo. Cuando necesite modificar el comportamiento de una función, solo tiene que modificar un bloque de código y su cambio surtirá efecto en todos los lugares donde haya realizado una llamada a esa función.

El uso de funciones hace que sus programas sean más fáciles de leer, y los buenos nombres de funciones resumen lo que hace cada parte de un programa. Leer una serie de llamadas a funciones le da una idea mucho más rápida de lo que hace un programa que leer una larga serie de bloques de código.

Las funciones también hacen que su código sea más fácil de probar y depurar. cuando el bulto del trabajo de su programa lo realiza un conjunto de funciones, cada una de las cuales tiene un trabajo específico, es mucho más fácil probar y mantener el código que ha escrito. Puede escribir un programa separado que llame a cada función y pruebe si cada función funciona en todas las situaciones que pueda encontrar. Cuando haga esto, puede estar seguro de que sus funciones funcionarán correctamente cada vez que las llame.

En el Capítulo 9 aprenderá a escribir clases. *Las clases* combinan funciones y datos en un paquete ordenado que se puede utilizar de manera flexible y eficiente.

9

Clases

La programación orientada a objetos es uno de los enfoques más efectivos para escribir software. En la programación orientada a objetos, escriba *clases* que representen cosas y situaciones del mundo real, y cree *objetos* basados en estas clases. Cuando escribes una clase, defines el comportamiento general que puede tener toda una categoría de objetos. Cuando crea objetos individuales de la clase, cada objeto se equipa automáticamente con el comportamiento general; luego puede darle a cada objeto las características únicas que desee. Se sorprenderá de lo bien que se pueden modelar situaciones del mundo real con la programación orientada a objetos.

Crear un objeto a partir de una clase se denomina creación de *instancias* y se trabaja con *instancias* de una clase. En este capítulo, escribirá clases y creará instancias de esas clases. Especificará el tipo de información que se puede almacenar en las instancias y definirá las acciones que se pueden realizar con estas instancias. También escribirá clases que amplían la funcionalidad de las clases existentes, por lo que

clases similares pueden compartir código de manera eficiente. Almacenará sus clases en módulos e importará clases escritas por otros programadores en sus propios archivos de programa.

Comprender la programación orientada a objetos te ayudará a ver el mundo como lo hace un programador. Lo ayudará a conocer realmente su código, no solo lo que sucede línea por línea, sino también los conceptos más importantes detrás de él. Conocer la lógica detrás de las clases lo entrenará para pensar lógicamente para que pueda escribir programas que aborden de manera efectiva casi cualquier problema que encuentre.

Las clases también te facilitan la vida a ti y a los demás programadores con los que trabajarás a medida que te enfrentas a desafíos cada vez más complejos. Cuando usted y otros programadores escriban código basado en el mismo tipo de lógica, podrán entender el trabajo de los demás. Sus programas tendrán sentido para muchos colaboradores, lo que permitirá que todos logren más.

Crear y usar una clase

Puedes modelar casi cualquier cosa usando clases. Comencemos escribiendo una clase simple, Perro, que represente un perro, no un perro en particular, sino cualquier perro. ¿Qué sabemos sobre la mayoría de los perros domésticos? Bueno, todos tienen un nombre y una edad. También sabemos que la mayoría de los perros se sientan y dan vueltas. Esas dos piezas de información (nombre y edad) y esos dos comportamientos (sentarse y darse la vuelta) irán en nuestra clase de perros porque son comunes a la mayoría de los perros. Esta clase le dirá a Python cómo hacer un objeto que represente a un perro. Después de escribir nuestra clase, la usaremos para crear instancias individuales, cada una de las cuales representa un perro específico.

Crear la clase de perro

Cada instancia creada a partir de la clase Perro almacenará un nombre y una edad, y le daremos a cada perro la capacidad de sentarse() y darse la vuelta():

```
dog.py u clase Perro:
    v """Un simple intento de modelar un perro."""

    w def __init__(yo, nombre, edad):
            """Inicialice los atributos de nombre y edad."""
    X       self.nombre = nombre
            self.edad = edad

    y def sentarse(auto):
            """Simule un perro sentado en respuesta a una orden"."""
            print(f"{self.name} ahora está sentado.")

        def roll_over(auto):
            """Simule darse la vuelta en respuesta a un comando."""
            print(f"{self.name} volcado!")
```

Hay mucho que notar aquí, pero no te preocupes. Verá esta estructura a lo largo de este capítulo y tendrá mucho tiempo para acostumbrarse a ella. en ti nosotros

definir una clase llamada Perro. Por convención, los nombres en mayúsculas se refieren a clases en Python. No hay paréntesis en la definición de la clase porque estamos creando esta clase desde cero. En v escribimos una cadena de documentación que describe lo que hace esta clase.

El método __init__()

Una función que es parte de una clase es un *método.* Todo lo que aprendió sobre las funciones también se aplica a los métodos; la única diferencia práctica por ahora es la forma en que llamaremos a los métodos. El método __init__() en w es un método especial que Python ejecuta automáticamente cada vez que creamos una nueva instancia basada en la clase Perro . Este método tiene dos guiones bajos iniciales y dos guiones bajos finales, una convención que ayuda a evitar que los nombres de métodos predeterminados de Python entren en conflicto con los nombres de sus métodos. Asegúrese de usar dos guiones bajos a cada lado de __init__(). Si usa solo uno en cada lado, el método no se llamará automáticamente cuando use su clase, lo que puede generar errores que son difíciles de identificar.

Definimos el método __init__() para que tenga tres parámetros: self, name y age. El parámetro self es obligatorio en la definición del método y debe aparecer antes que los demás parámetros. Debe incluirse en la definición porque cuando Python llame a este método más adelante (para crear una instancia de Dog), la llamada al método pasará automáticamente el argumento self . Cada llamada de método asociada con una instancia pasa automáticamente a self, que es una referencia a la instancia misma; le da a la instancia individual acceso a los atributos y métodos de la clase. Cuando creamos una instancia de Dog, Python llamará al método __init__() de la clase Dog . Pasaremos Dog()

un nombre y una edad como argumentos; self se pasa automáticamente, por lo que no necesitamos pasarlo. Siempre que queramos crear una instancia de la clase Perro , proporcionaremos valores solo para los dos últimos parámetros, nombre y edad.

Las dos variables definidas en x tienen cada una el prefijo self. Cualquier variable con el prefijo self está disponible para todos los métodos de la clase, y también podremos acceder a estas variables a través de cualquier instancia creada a partir de la clase. La línea self.name = name toma el valor asociado con el nombre del parámetro y lo asigna al nombre de la variable , que luego se adjunta a la instancia que se está creando. El mismo proceso ocurre con self.edad = edad. Las variables a las que se puede acceder a través de instancias como esta se denominan *atributos.*

La clase Dog tiene otros dos métodos definidos: sit() y roll_over() y. Debido a que estos métodos no necesitan información adicional para ejecutarse, solo los definimos para que tengan un parámetro, self. Las instancias que creemos más tarde tendrán acceso a estos métodos. En otras palabras, podrán sentarse y darse la vuelta. Por ahora, sit() y roll_over() no hacen mucho. Simplemente imprimen un mensaje que dice que el perro está sentado o volteándose. Pero el concepto se puede extender a situaciones realistas: si esta clase fuera parte de un juego de computadora real, estos métodos contendrían un código para hacer que un perro animado se siente y se dé la vuelta. Si esta clase se escribió para controlar un robot, estos métodos dirigirían los movimientos que hacen que un perro robótico se siente y se dé la vuelta.

Crear una instancia de una clase

Piense en una clase como un conjunto de instrucciones sobre cómo crear una instancia. La clase Dog es un conjunto de instrucciones que le dice a Python cómo crear instancias individuales que representen perros específicos.

Hagamos una instancia que represente a un perro específico:

```
perro de clase:
    --recorte--

u mi_perro = Perro('Willie', 6)

v print(f"El nombre de mi perro es {mi_perro.nombre}").
w print(f"Mi perro tiene {mi_perro.edad} años")
```

La clase Dog que estamos usando aquí es la que acabamos de escribir en el ejemplo anterior. En u le decimos a Python que cree un perro cuyo nombre sea 'Willie' y cuya edad sea 6. Cuando Python lee esta línea, llama al método __init__() en Dog con los argumentos 'Willie' y 6. El método __init__() crea una instancia que representa a este perro en particular y establece los atributos de nombre y edad usando los valores que proporcionamos. Python luego devuelve una instancia que representa a este perro. Asignamos esa instancia a la variable mi_perro. La convención de nomenclatura es útil aquí: generalmente podemos suponer que un nombre en mayúsculas como Perro

se refiere a una clase, y un nombre en minúsculas como my_dog se refiere a una única instancia creada a partir de una clase.

Accediendo a los Atributos

Para acceder a los atributos de una instancia, utiliza la notación de puntos. En v accedemos al valor del nombre del atributo de mi_perro escribiendo:

```
mi_perro.nombre
```

La notación de puntos se usa a menudo en Python. Esta sintaxis demuestra cómo Python encuentra el valor de un atributo. Aquí Python mira la instancia my_dog
y luego encuentra el nombre del atributo asociado con my_dog. Este es el mismo atributo al que se hace referencia como self.name en la clase Dog. En w usamos el mismo enfoque para trabajar con el atributo edad.

El resultado es un resumen de lo que sabemos sobre my_dog:

```
El nombre de mi perro es Willie.
Mi perro tiene 6 años.
```

Métodos de llamada

Después de crear una instancia de la clase Dog, podemos usar la notación de puntos para llamar a cualquier método definido en Dog. Hagamos que nuestro perro se siente y se dé la vuelta:

```
perro de clase:
    --recorte--
```

```
mi_perro = Perro('Willie', 6)
mi_perro.sentarse()
mi_perro.roll_over()
```

Para llamar a un método, proporcione el nombre de la instancia (en este caso, mi_perro) y el método al que desea llamar, separados por un punto. Cuando Python lee my_dog.sit(), busca el método sit() en la clase Dog y ejecuta ese código. Python interpreta la línea my_dog.roll_over() de la misma manera.

Ahora Willie hace lo que le decimos:

```
Willie ahora está sentado.
¡Willie se dio la vuelta!
```

Esta sintaxis es bastante útil. Cuando a los atributos y métodos se les han dado nombres descriptivos apropiados como nombre, edad, sit() y roll_over(), podemos inferir fácilmente qué se supone que debe hacer un bloque de código, incluso uno que nunca hayamos visto antes.

Creación de varias instancias

Puede crear tantas instancias de una clase como necesite. Vamos a crear un segundo perro llamado tu_perro:

```
perro de clase:
    --recorte--

mi_perro = Perro('Willie', 6)
tu_perro = Perro('Lucy', 3)

print(f"El nombre de mi perro es {mi_perro.nombre}").
print(f"Mi perro tiene {mi_perro.edad} años")
mi_perro.sentarse()

print(f"\nEl nombre de tu perro es {tu_perro.nombre}").
print(f"Tu perro tiene {tu_perro.edad} años").
tu_perro.siéntate()
```

En este ejemplo, creamos un perro llamado Willie y un perro llamado Lucy. Cada perro es una instancia separada con su propio conjunto de atributos, capaz de realizar el mismo conjunto de acciones:

```
El nombre de mi perro es Willie.
Mi perro tiene 6 años.
Willie ahora está sentado.

Tu perro se llama Lucy.
Tu perro tiene 3 años.
Lucy ahora está sentada.
```

Incluso si usáramos el mismo nombre y edad para el segundo perro, Python aún crearía una instancia separada de la clase Perro . Puedes hacer

tantas instancias de una clase como necesite, siempre que asigne a cada instancia un nombre de variable único o que ocupe un lugar único en una lista o diccionario.

Inténtalo tú mismo

9-1. Restaurante: Haz una clase llamada Restaurante. El método __init__() para Restaurante debe almacenar dos atributos: un nombre_restaurante y un tipo_cocina . Haz un método llamado describe_restaurant() que imprima estas dos piezas de información, y un método llamado open_restaurant() que imprima un mensaje que indique que el restaurante está abierto.

Cree una instancia llamada restaurante de su clase. Imprima los dos atributos individualmente y luego llame a ambos métodos.

9-2. Tres Restaurantes: Comience con su clase del Ejercicio 9-1. Cree tres instancias diferentes de la clase y llame a describe_restaurant() para cada instancia.

9-3. Usuarios: Haz una clase llamada Usuario. Crea dos atributos llamados first_name y last_name, y luego cree varios otros atributos que normalmente se almacenan en un perfil de usuario. Cree un método llamado describe_user() que imprima un resumen de la información del usuario. Cree otro método llamado greeting_user() que imprima un saludo personalizado para el usuario.

Cree varias instancias que representen a diferentes usuarios y llame a ambos métodos para cada usuario.

Trabajar con clases e instancias

Puede usar clases para representar muchas situaciones del mundo real. Una vez que escriba una clase, pasará la mayor parte de su tiempo trabajando con instancias creadas a partir de esa clase. Una de las primeras tareas que querrá hacer es modificar los atributos asociados con una instancia en particular. Puede modificar los atributos de una instancia directamente o escribir métodos que actualicen los atributos de formas específicas.

la clase de coche

Escribamos una nueva clase que represente un automóvil. Nuestra clase almacenará información sobre el tipo de automóvil con el que estamos trabajando y tendrá un método que resuma esta información:

coche.py

```
Coche de clase:
    """Un simple intento de representar un coche."""

u def __init__(auto, marca, modelo, año):
        """Inicialice los atributos para describir un automóvil".""
        self.hacer = hacer
```

```
        self.modelo = modelo
        self.año = año

v def get_descriptive_name(self):
        """Retorna un nombre descriptivo con un formato ordenado."""
        long_name = f"{self.year} {self.manufacturer} {self.model}"
        devuelve nombre_largo.título()

w mi_auto_nuevo = Auto('audi', 'a4', 2019)
  print(my_new_car.get_descriptive_name())
```

En u en la clase Car , definimos el método __init__() con el self parámetro primero, tal como lo hicimos antes con nuestra clase Dog . También le damos otros tres parámetros: marca, modelo y año. El método __init__() toma estos parámetros y los asigna a los atributos que se asociarán con las instancias creadas a partir de esta clase. Cuando fabricamos un coche nuevo instancia, necesitaremos especificar una marca, modelo y año para nuestra instancia.

En v definimos un método llamado get_descriptive_name() que pone el año, la marca y el modelo de un automóvil en una cadena que describe claramente el automóvil. Esto nos evitará tener que imprimir el valor de cada atributo individualmente. Para trabajar con los valores de los atributos en este método, usamos self.make, self.model y self.year. En w creamos una instancia de la clase Car y la asignamos a la variable my_new_car. Luego llamamos a get_descriptive_name() para mostrar qué tipo de automóvil tenemos:

```
2019 Audi A4
```

Para hacer la clase más interesante, agreguemos un atributo que cambie tiempo extraordinario. Agregaremos un atributo que almacene el kilometraje total del automóvil.

Establecer un valor predeterminado para un atributo

Cuando se crea una instancia, los atributos se pueden definir sin pasarlos como parámetros. Estos atributos se pueden definir en __init__() método, donde se les asigna un valor por defecto.

Agreguemos un atributo llamado odometer_reading que siempre comienza con un valor de 0. También agregaremos un método read_odometer() que nos ayuda a leer el odómetro de cada automóvil:

```
Coche de clase:

    def __init__(auto, marca, modelo, año):
        """Inicialice los atributos para describir un automóvil".""
        self.hacer = hacer
        self.modelo = modelo
        self.año = año
en      self.odometer_reading = 0

    def get_descriptive_name(self):
        --recorte--
```

```
v def read_odometer(self):
        """Imprima una declaración que muestre el kilometraje del automóvil"."""
        print(f"Este auto tiene {self.odometer_reading} millas").

mi_auto_nuevo = Auto('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
mi_auto_nuevo.leer_odómetro()
```

Esta vez, cuando Python llama al método __init__() para crear una nueva instancia, almacena los valores de marca, modelo y año como atributos, como lo hizo en el ejemplo anterior. Luego, Python crea un nuevo atributo llamado odometer_reading y establece su valor inicial en 0 u. También tenemos un nuevo método llamado read_odometer() en v que facilita la lectura del kilometraje de un automóvil.

Nuestro coche arranca con un kilometraje de 0:

```
2019 Audi A4
Este coche tiene 0 millas en él.
```

No se venden muchos autos con exactamente 0 millas en el odómetro, por lo que necesitamos una forma de cambiar el valor de este atributo.

Modificación de valores de atributos

Puede cambiar el valor de un atributo de tres maneras: puede cambiar el valor directamente a través de una instancia, establecer el valor a través de un método o incrementar el valor (agregarle una cierta cantidad) a través de un método. Veamos cada uno de estos enfoques.

Modificar el valor de un atributo directamente

La forma más sencilla de modificar el valor de un atributo es acceder al atributo directamente a través de una instancia. Aquí establecemos la lectura del odómetro a 23 directamente:

```
Coche de clase:
    --recorte--

mi_auto_nuevo = Auto('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())

u my_new_car.odometer_reading = 23
mi_auto_nuevo.leer_odómetro()
```

En u usamos la notación de puntos para acceder al atributo de lectura del odómetro del automóvil y establecer su valor directamente. Esta línea le dice a Python que tome la instancia my_new_car, encuentre el atributo odometer_reading asociado con él y establezca el valor de ese atributo en 23:

```
2019 Audi A4
Este coche tiene 23 millas en él.
```

A veces querrás acceder a atributos directamente como este, pero otras veces querrá escribir un método que actualice el valor por usted.

Modificar el valor de un atributo a través de un método

Puede ser útil tener métodos que actualicen ciertos atributos por usted. En lugar de acceder al atributo directamente, pasa el nuevo valor a un método que maneja la actualización internamente.

Aquí hay un ejemplo que muestra un método llamado update_odometer():

```
Coche de clase:
    --recorte--

u def update_odometer(self, kilometraje):
        """Ajuste la lectura del cuentakilómetros al valor indicado."""
        self.odometer_reading = kilometraje

mi_auto_nuevo = Auto('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())

v my_new_car.update_odometer(23)
mi_auto_nuevo.leer_odómetro()
```

La única modificación a Car es la adición de update_odometer() en u. Este método toma un valor de kilometraje y lo asigna a self.odometer_reading. En v, llamamos a update_odometer() y le damos 23 como argumento (correspondiente al parámetro de kilometraje en la definición del método). Establece la lectura del odómetro en 23 y read_odometer() imprime la lectura:

```
2019 Audi A4
Este coche tiene 23 millas en él.
```

Podemos extender el método update_odometer() para hacer un trabajo adicional cada vez que se modifica la lectura del odómetro. Agreguemos un poco de lógica para asegurarnos de que nadie intente hacer retroceder la lectura del odómetro:

```
Coche de clase:
    --recorte--

    def update_odometer(self, kilometraje):
        """
        Ajuste la lectura del odómetro al valor dado.
        Rechace el cambio si intenta hacer retroceder el odómetro.
        """
en      si kilometraje >= self.odometer_reading:
            self.odometer_reading = kilometraje
        demás:
en          print("¡No puedes retroceder un odómetro!")
```

Ahora update_odometer() verifica que la nueva lectura tenga sentido antes de modificar el atributo. Si el nuevo kilometraje, kilometraje, es mayor o igual

al kilometraje existente, self.odometer_reading, puede actualizar la lectura del odómetro al nuevo kilometraje u. Si el nuevo kilometraje es menor que el kilometraje existente, recibirá una advertencia de que no puede retroceder un odómetro v.

Incrementar el valor de un atributo a través de un método

A veces querrá incrementar el valor de un atributo en cierta cantidad en lugar de establecer un valor completamente nuevo. Digamos que compramos un automóvil usado y acumulamos 100 millas entre el momento en que lo compramos y el momento en que lo registramos.
Aquí hay un método que nos permite pasar esta cantidad incremental y agregar ese valor a la lectura del odómetro:

```
Coche de clase:
    --recorte--

    def update_odometer(self, kilometraje):
        --recorte--

u def increment_odometer(self, miles):
        """Agregue la cantidad dada a la lectura del odómetro."""
        self.odometer_reading += millas

v my_used_car = Car('subaru', 'outback', 2015)
  print(my_used_car.get_descriptive_name())

w my_used_car.update_odometer(23_500)
  mi_coche_usado.leer_odómetro()

x my_used_car.increment_odometer(100)
  mi_coche_usado.leer_odómetro()
```

El nuevo método increment_odometer() en u toma un número de millas, y agrega este valor a self.odometer_reading. En v creamos un auto usado, my_used_car. Establecemos su cuentakilómetros en 23 500 llamando a update_odometer() y pasándole 23_500 en w. En x llamamos a increment_odometer() y le pasamos 100 para sumar las 100 millas que recorrimos entre comprar el auto y registrarlo:

```
2015 Subaru Outback
Este auto tiene 23500 millas.
Este auto tiene 23600 millas.
```

Puede modificar fácilmente este método para rechazar incrementos negativos para que no uno usa esta función para hacer retroceder un odómetro.

Puede usar métodos como este para controlar cómo los usuarios de su programa actualizan los valores, como la lectura del odómetro, pero cualquier persona con acceso al programa puede establecer la lectura del odómetro en cualquier valor accediendo directamente al atributo. La seguridad eficaz exige una atención extrema a los detalles además de las comprobaciones básicas como las que se muestran aquí.

Inténtalo tú mismo

9-4. Número servido: comience con su programa del ejercicio 9-1 (página 162). Agregue un atributo llamado número_servido con un valor predeterminado de 0. Cree una instancia llamada restaurante a partir de esta clase. Imprime el número de clientes que ha servido el restaurante y luego cambia este valor e imprímelo de nuevo.

Agregue un método llamado set_number_served() que le permita establecer la cantidad de clientes que han sido atendidos. Llame a este método con un nuevo número e imprima el valor nuevamente.

Agregue un método llamado increment_number_served() que le permita incrementar la cantidad de clientes que han sido atendidos. Llame a este método con cualquier número que desee que pueda representar cuántos clientes se atendieron, digamos, en un día hábil.

9-5. Intentos de inicio de sesión: agregue un atributo llamado intentos de inicio de sesión a su usuario clase del ejercicio 9-3 (página 162). Escriba un método llamado increment_login _attempts() que incrementa el valor de login_attempts en 1. Escribe otro método llamado reset_login_attempts() que restablece el valor de login_attempts a 0.

Crea una instancia de la clase User y llama a increment_login_attempts() varias veces. Imprima el valor de login_attempts para asegurarse de que se incrementó correctamente y luego llame a reset_login_attempts(). Imprima login_attempts nuevamente para asegurarse de que se restableció a 0.

Herencia

No siempre tienes que empezar desde cero al escribir una clase. Si la clase que está escribiendo es una versión especializada de otra clase que escribió, puede usar la *herencia.* Cuando una clase *hereda* de otra, adquiere los atributos y métodos de la primera clase. La clase original se llama *clase padre* y la nueva clase es la *clase hija.* La clase secundaria puede heredar cualquiera o todos los atributos y métodos de su clase principal, pero también es libre de definir sus propios atributos y métodos nuevos.

El método __init__() para una clase secundaria

Cuando está escribiendo una nueva clase basada en una clase existente, a menudo querrá llamar al método __init__() desde la clase principal. Esto inicializará todos los atributos que se definieron en el método principal __init__() y los hará disponibles en la clase secundaria.

Como ejemplo, modelemos un coche eléctrico. Un automóvil eléctrico es solo un tipo específico de automóvil, por lo que podemos basar nuestra nueva clase ElectricCar en la clase Car que escribimos anteriormente. Entonces solo tendremos que escribir código para los atributos y el comportamiento específico de los autos eléctricos.

Comencemos por hacer una versión simple de la clase ElectricCar , que hace todo lo que hace la clase Car :

```
electric_car.py u class Car:
    """Un simple intento de representar un coche."""

    def __init__(auto, marca, modelo, año):
        self.hacer = hacer
        self.modelo = modelo
        self.año = año
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.manufacturer} {self.model}"
        devuelve nombre_largo.título()

    def read_odometer(self):
        print(f"Este auto tiene {self.odometer_reading} millas").

    def update_odometer(self, kilometraje):
        si kilometraje >= self.odometer_reading:
            self.odometer_reading = kilometraje
        demás:
            print("¡No puedes retroceder un odómetro!")

    def increment_odometer(self, millas):
        self.odometer_reading += millas

v clase ElectricCar(Coche):
    """Representan aspectos de un coche, específicos de los vehículos eléctricos."""

w def __init__(auto, marca, modelo, año):
        """Inicializar atributos de la clase padre."""
X       super().__init__(marca, modelo, año)


y my_tesla = ElectricCar('tesla', 'modelo s', 2019)
  imprimir (my_tesla.get_descriptive_name())
```

En u comenzamos con Car. Cuando crea una clase secundaria, la clase principal debe ser parte del archivo actual y debe aparecer antes que la clase secundaria en el archivo. En v definimos la clase secundaria, ElectricCar. El nombre de la clase principal debe incluirse entre paréntesis en la definición de una clase secundaria. El método __init__() en w toma la información requerida para hacer un Car instancia.

La función super() en x es una función especial que le permite llamar a un método de la clase principal. Esta línea le dice a Python que llame a __init__() de Car, que otorga a una instancia de ElectricCar todos los atributos definidos en ese método. El nombre *super* proviene de una convención de llamar a la clase padre *superclase* y a la clase hija *subclase.*

Probamos si la herencia funciona correctamente al intentar crear un automóvil eléctrico con el mismo tipo de información que proporcionaríamos al fabricar un automóvil normal. En y creamos una instancia de la clase ElectricCar y la asignamos a my_tesla. Esta línea llama al método __init__() definido en ElectricCar, que a su vez le dice a Python que llame al método __init__() definido en la clase padre Car. Proporcionamos los argumentos 'tesla', 'model s' y 2019.

Aparte de __init__(), todavía no hay atributos o métodos que sean particulares de un automóvil eléctrico. En este punto, solo nos estamos asegurando de que el automóvil eléctrico tenga los comportamientos de automóvil apropiados:

```
Modelo S de Tesla 2019
```

La instancia de ElectricCar funciona como una instancia de Car, así que ahora puede comenzar definiendo atributos y métodos específicos para los autos eléctricos.

Definición de atributos y métodos para la clase secundaria

Una vez que tenga una clase secundaria que herede de una clase principal, puede agregar los nuevos atributos y métodos necesarios para diferenciar la clase secundaria de la clase principal.

Agreguemos un atributo que sea específico para los autos eléctricos (una batería, por ejemplo) y un método para informar sobre este atributo. Guardaremos el tamaño de la batería y escribiremos un método que imprima una descripción de la batería:

```
Coche de clase:
    --recorte--

clase ElectricCar(Coche):
    """Representan aspectos de un coche, específicos de los vehículos eléctricos."""

    def __init__(auto, marca, modelo, año):
        """
        Inicializar atributos de la clase padre.
        A continuación, inicialice los atributos específicos de un coche eléctrico.
        """
        super().__init__(marca, modelo, año)
en      self.battery_size = 75

v def describe_batería(uno mismo):
        """Imprima una declaración que describa el tamaño de la batería."""
        print(f"Este auto tiene una batería de {self.battery_size}-kWh").

my_tesla = ElectricCar('tesla', 'modelo s', 2019)
imprimir (my_tesla.get_descriptive_name())
my_tesla.describe_battery()
```

En u agregamos un nuevo atributo self.battery_size y establecemos su valor inicial en, digamos, 75. Este atributo se asociará con todas las instancias creadas a partir de la clase ElectricCar pero no se asociará con ninguna instancia de Car. Nosotros también

agrega un método llamado describe_battery() que imprime información sobre la batería en v. Cuando llamamos a este método, obtenemos una descripción que es claramente específica de un automóvil eléctrico:

```
Tesla Model S 2019
Este automóvil tiene una batería de 75 kWh.
```

No hay límite en cuánto puede especializarse en la clase ElectricCar . Puede agregar tantos atributos y métodos como necesite para modelar un automóvil eléctrico con el grado de precisión que necesite. Un atributo o método que podría pertenecer a cualquier automóvil, en lugar de uno que sea específico de un automóvil eléctrico, debe agregarse a la clase Car en lugar de a la clase ElectricCar . Entonces, cualquiera que use la clase Car también tendrá esa funcionalidad disponible, y la clase ElectricCar solo contendrá el código para la información y el comportamiento específico de los vehículos eléctricos.

Anulación de métodos de la clase principal

Puede anular cualquier método de la clase principal que no se ajuste a lo que intenta modelar con la clase secundaria. Para hacer esto, defina un método en la clase secundaria con el mismo nombre que el método que desea anular en la clase principal. Python ignorará el método de la clase principal y solo prestará atención al método que defina en la clase secundaria.

Digamos que la clase Car tenía un método llamado fill_gas_tank(). Este método no tiene sentido para un vehículo totalmente eléctrico, por lo que es posible que desee anular este método. Aquí hay una forma de hacerlo:

```
clase ElectricCar(Coche):
    --recorte--

    def llenar_tanque_de_gas(auto):
        """Los autos eléctricos no tienen tanques de gasolina"."""
        print("¡Este auto no necesita tanque de gasolina!")
```

Ahora, si alguien intenta llamar a fill_gas_tank() con un automóvil eléctrico, Python ignorará el método fill_gas_tank() en Car y ejecutará este código en su lugar. Cuando usa la herencia, puede hacer que sus clases secundarias conserven lo que necesita y anule todo lo que no necesite de la clase principal.

Instancias como atributos

Al modelar algo del mundo real en código, es posible que descubra que está agregando más y más detalles a una clase. Descubrirá que tiene una lista creciente de atributos y métodos y que sus archivos son cada vez más largos. En estas situaciones, es posible que reconozca que parte de una clase se puede escribir como una clase separada. Puede dividir su clase grande en clases más pequeñas que trabajen juntas.

Por ejemplo, si continuamos agregando detalles a la clase ElectricCar , podemos notar que estamos agregando muchos atributos y métodos específicos para

la batería del coche. Cuando vemos que esto sucede, podemos detener y mover esos atributos y métodos a una clase separada llamada Batería. Entonces podemos usar una instancia de Batería como atributo en la clase ElectricCar :

```
Coche de clase:
    --recorte--

batería de clase u:
    """Un simple intento de modelar una batería para un coche eléctrico."""

v def __init__(self, tamaño_batería=75):
        """Inicialice los atributos de la batería."""
        self.battery_size = batería_tamaño

w def describe_batería(uno mismo):
        """Imprima una declaración que describa el tamaño de la batería."""
        print(f"Este auto tiene una batería de {self.battery_size}-kWh").


clase ElectricCar(Coche):
    """Representan aspectos de un coche, específicos de los vehículos eléctricos."""

    def __init__(auto, marca, modelo, año):
        """
        Inicializar atributos de la clase padre.
        A continuación, inicialice los atributos específicos de un coche eléctrico.
        """
        super().__init__(marca, modelo, año)
x       self.bateria = Bateria()


my_tesla = ElectricCar('tesla', 'modelo s', 2019)

imprimir (my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
```

En u definimos una nueva clase llamada Batería que no hereda de ninguna otra clase El método __init__() en v tiene un parámetro, battery_size, además de self. Este es un parámetro opcional que establece el tamaño de la batería en 75 si no se proporciona ningún valor. El método describe_battery() también se ha movido a esta clase w.

En la clase ElectricCar , ahora agregamos un atributo llamado self.battery x. Esta línea le dice a Python que cree una nueva instancia de Batería (con un tamaño predeterminado de 75, porque no estamos especificando un valor) y asigne esa instancia al atributo self.battery. Esto sucederá cada vez que se llame al método __init__() ; cualquier instancia de ElectricCar ahora tendrá una instancia de Batería creada automáticamente.

Creamos un coche eléctrico y lo asignamos a la variable my_tesla. Cuando queremos describir la batería, necesitamos trabajar con la batería del automóvil. atributo:

```
my_tesla.battery.describe_battery()
```

Esta línea le dice a Python que mire la instancia my_tesla, encuentre su batería atributo y llame al método describe_battery() que está asociado con la instancia de batería almacenada en el atributo.

El resultado es idéntico al que vimos anteriormente:

```
Tesla Model S 2019 Este
automóvil tiene una batería de 75 kWh.
```

Esto parece mucho trabajo extra, pero ahora podemos describir la batería con tanto detalle como queramos sin saturar la clase ElectricCar . Agreguemos otro método a Batería que informe el alcance del automóvil según el tamaño de la batería:

```
Coche de clase:
    --recorte--

batería de clase:
    --recorte--

u def get_range(self):
        """Imprime una declaración sobre el alcance que proporciona esta batería."""
        si self.battery_size == 75:
            rango = 260
        elif self.battery_size == 100:
            rango = 315

        print(f"Este auto puede recorrer alrededor de {rango} millas con una carga completa").

clase ElectricCar(Coche):
    --recorte--

my_tesla = ElectricCar('tesla', 'modelo s', 2019)
imprimir (my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
v my_tesla.battery.get_range()
```

El nuevo método get_range() en u realiza un análisis simple. Si la capacidad de la batería es de 75 kWh, get_range() establece el alcance en 260 millas, y si la capacidad es de 100 kWh, establece el alcance en 315 millas. Luego informa este valor. Cuando queramos usar este método, nuevamente tenemos que llamarlo a través del atributo de batería del automóvil en v.

La salida nos dice la autonomía del coche en función del tamaño de la batería:

```
Tesla Model S 2019 Este
automóvil tiene una batería de 75 kWh.
Este coche puede recorrer unas 260 millas con una carga completa.
```

Modelado de objetos del mundo real

A medida que comience a modelar cosas más complicadas, como automóviles eléctricos, se enfrentará a preguntas interesantes. ¿La autonomía de un coche eléctrico es una propiedad de la batería o del coche? Si solo estamos describiendo un automóvil, probablemente esté bien mantener la asociación del método get_range() con la batería clase. Pero si estamos describiendo toda la línea de automóviles de un fabricante, probablemente queramos mover get_range() a la clase ElectricCar . El método get_range() aún verificaría el tamaño de la batería antes de determinar el rango, pero informaría un rango específico para el tipo de automóvil con el que está asociado. Alternativamente, podríamos mantener la asociación del método get_range() con la batería pero pasarle un parámetro como car_model. El método get_range() luego informaría un rango basado en el tamaño de la batería y el modelo de automóvil.

Esto lo lleva a un punto interesante en su crecimiento como programador. Cuando lucha con preguntas como estas, está pensando en un nivel lógico superior en lugar de un nivel centrado en la sintaxis. No estás pensando en Python, sino en cómo representar el mundo real en código. Cuando llegue a este punto, se dará cuenta de que a menudo no hay enfoques correctos o incorrectos para modelar situaciones del mundo real. Algunos enfoques son más eficientes que otros, pero se necesita práctica para encontrar las representaciones más eficientes. Si tu código funciona como quieres, ¡lo estás haciendo bien! No se desanime si descubre que está fragmentando sus clases y reescribiéndolas varias veces utilizando diferentes enfoques. En la búsqueda de escribir código preciso y eficiente, todo el mundo pasa por este proceso.

Inténtalo tú mismo

9-6. Puesto de helados: Un puesto de helados es un tipo específico de restaurante. Escriba una clase llamada IceCreamStand que herede de la clase Restaurante que escribió en el Ejercicio 9-1 (página 162) o el Ejercicio 9-4 (página 167). Cualquiera de las versiones de la clase funcionará; simplemente elige el que más te guste. Agregue un atributo llamado sabores que almacene una lista de sabores de helado. Escribe un método que muestre estos sabores. Cree una instancia de IceCreamStand y llame a este método.

9-7. Admin: Un administrador es un tipo especial de usuario. Escriba una clase llamada Admin que herede de la clase Usuario que escribió en el Ejercicio 9-3 (página 162) o el Ejercicio 9-5 (página 167). Agregue un atributo, privilegios, que almacene una lista de cadenas como "puede agregar una publicación", "puede eliminar una publicación", "puede prohibir a un usuario", etc. Escriba un método llamado show_privileges() que enumere el conjunto de privilegios del administrador. Cree una instancia de Admin y llame a su método.

9-8. Privilegios: Escriba una clase de Privilegios separada . La clase debe tener un atributo, privilegios, que almacene una lista de cadenas como se describe en el ejercicio 9-7. Mueva el método show_privileges() a esta clase. Cree una instancia de Privilegios como un atributo en la clase Admin . Cree una nueva instancia de Admin y use su método para mostrar sus privilegios.

(continuado)

9-9. Actualización de batería: use la versión final de electric_car.py de esta sección. Agregue un método a la clase Batería llamado upgrade_battery(). Este método debería verificar el tamaño de la batería y establecer la capacidad en 100 si aún no lo está. Haga un automóvil eléctrico con un tamaño de batería predeterminado, llame a get_range() una vez y luego llame a get_range() una segunda vez después de actualizar la batería. Debería ver un aumento en el alcance del automóvil.

Importación de clases

A medida que agrega más funcionalidad a sus clases, sus archivos pueden alargarse, incluso cuando usa la herencia correctamente. De acuerdo con la filosofía general de Python, querrá mantener sus archivos lo más ordenados posible. Para ayudar, Python le permite almacenar clases en módulos y luego importar las clases que necesita en su programa principal.

Importación de una sola clase

Vamos a crear un módulo que contenga solo la clase Car . Esto trae a colación un sutil problema de nomenclatura: ya tenemos un archivo llamado *car.py* en este capítulo, pero este módulo debería llamarse *car.py* porque contiene código que representa un auto. Resolveremos este problema de nombres almacenando la clase Car en un módulo llamado *car.py,* reemplazando el archivo *car.py* que estábamos usando anteriormente. De ahora en adelante, cualquier programa que use este módulo necesitará un nombre de archivo más específico, como *my_car.py.* Aquí está *car.py* con solo el código de la clase Car:

```
car.py u """Una clase que se puede usar para representar un automóvil".""

Coche de clase:
    """Un simple intento de representar un coche."""

    def __init__(auto, marca, modelo, año):
        """Inicialice los atributos para describir un automóvil".""
        self.hacer = hacer
        self.modelo = modelo
        self.año = año
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Retorna un nombre descriptivo con un formato ordenado."""
        long_name = f"{self.year} {self.manufacturer} {self.model}"
        devuelve nombre_largo.título()

    def read_odometer(self):
        """Imprima una declaración que muestre el kilometraje del automóvil".""
        print(f"Este auto tiene {self.odometer_reading} millas").
```

```
    def update_odometer(self, kilometraje):
        """
        Ajuste la lectura del odómetro al valor dado.
        Rechace el cambio si intenta hacer retroceder el odómetro.
        """
        si kilometraje >= self.odometer_reading:
            self.odometer_reading = kilometraje
        demás:
            print("¡No puedes retroceder un odómetro!")

    def increment_odometer(self, millas):
        """Agregue la cantidad dada a la lectura del odómetro."""
        self.odometer_reading += millas
```

En u incluimos una cadena de documentación a nivel de módulo que describe brevemente el contenido de este módulo. Debe escribir una cadena de documentación para cada módulo que crear.

Ahora creamos un archivo separado llamado *my_car.py.* Este archivo importará la clase Car y luego creará una instancia de esa clase:

```
my_car.py u from car import Car

mi_auto_nuevo = Auto('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
mi_auto_nuevo.leer_odómetro()
```

La declaración de importación en u le dice a Python que abra el módulo del automóvil e importe la clase Car. Ahora podemos usar la clase Car como si estuviera definida en este archivo. La salida es la misma que vimos antes:

```
Audi A4 2019
Este auto tiene 23 millas.
```

Importar clases es una forma efectiva de programar. Imagine cuánto tiempo sería este archivo de programa si se incluyera toda la clase de automóvil . Cuando, en cambio, mueve la clase a un módulo e importa el módulo, aún obtiene la misma funcionalidad, pero mantiene su archivo de programa principal limpio y fácil de leer. También almacena la mayor parte de la lógica en archivos separados; una vez que sus clases funcionen como desea, puede dejar esos archivos solos y concentrarse en la lógica de nivel superior de su programa principal.

Almacenamiento de varias clases en un módulo

Puede almacenar tantas clases como necesite en un solo módulo, aunque cada clase en un módulo debe estar relacionada de alguna manera. Las clases Batería y ElectricCar ayudan a representar automóviles, así que agréguelos al módulo *car.py.*

coche.py

```
"""Un conjunto de clases que se utiliza para representar automóviles de gasolina y eléctricos"."""

Coche de clase:
    --recorte--

class Batería:
    """Un simple intento de modelar una batería para un coche eléctrico."""

    def __init__(self, tamaño_batería=70): """Inicializa
        los atributos de la batería.""" self.tamaño_batería =
        tamaño_batería

    def describe_batería(uno mismo):
        """Imprime una declaración que describa el tamaño de la batería"."""
        print(f"Este automóvil tiene una batería de {self.battery_size}-kWh").

    def get_range(self):
        """Imprime una declaración sobre el alcance que proporciona esta batería.""" if
        self.battery_size == 75: range = 260 elif self.battery_size == 100: range = 315




        print(f"Este auto puede recorrer alrededor de {rango} millas con una carga completa").

clase ElectricCar(Coche):
    """Modela aspectos de un coche, específicos para vehículos eléctricos."""

    def __init__(auto, marca, modelo, año):
        """
        Inicializar atributos de la clase padre.
        A continuación, inicialice los atributos específicos de un coche eléctrico.
        """
        super().__init__(marca, modelo, año) self.battery
        = Batería()
```

Ahora podemos crear un nuevo archivo llamado *my_electric_car.py,* importar la clase ElectricCar y hacer un auto eléctrico:

mi_electrico
_coche.py

```
de coche de importación ElectricCar

my_tesla = ElectricCar('tesla', 'modelo s', 2019)

print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
my_tesla.battery.get_range()
```

Esto tiene el mismo resultado que vimos antes, aunque la mayor parte de la lógica está oculta en un módulo:

```
Tesla Model S 2019 Este
automóvil tiene una batería de 75 kWh.
Este coche puede recorrer unas 260 millas con una carga completa.
```

Importación de varias clases desde un módulo

Puede importar tantas clases como necesite en un archivo de programa. Si queremos hacer un auto normal y un auto eléctrico en el mismo archivo, necesitamos importar ambas clases, Car y ElectricCar:

```
my_cars.py u from car import Car, ElectricCar

v mi_escarabajo = Coche('volkswagen', 'escarabajo', 2019)
    imprimir(mi_escarabajo.obtener_nombre_descriptivo())

w my_tesla = ElectricCar('tesla', 'roadster', 2019)
    imprimir (my_tesla.get_descriptive_name())
```

Importa varias clases de un módulo separando cada clase con una coma u. Una vez que haya importado las clases necesarias, puede crear tantas instancias de cada clase como necesite.

En este ejemplo, hacemos un Volkswagen Beetle normal en v y uno eléctrico. tric Tesla Roadster en w:

```
Volkswagen Escarabajo 2019
Descapotable Tesla 2019
```

Importación de un módulo completo

También puede importar un módulo completo y luego acceder a las clases que necesita usando la notación de puntos. Este enfoque es simple y da como resultado un código que es fácil de leer. Debido a que cada llamada que crea una instancia de una clase incluye el nombre del módulo, no tendrá conflictos de nombres con los nombres utilizados en el archivo actual.

Esto es lo que parece importar todo el módulo del automóvil y luego crear un coche normal y un coche eléctrico:

```
my_cars.py u coche de importación

v mi_escarabajo = coche.Coche('volkswagen', 'escarabajo', 2019)
    imprimir(mi_escarabajo.obtener_nombre_descriptivo())

w my_tesla = auto.ElectricCar('tesla', 'roadster', 2019)
    imprimir (my_tesla.get_descriptive_name())
```

En u importamos todo el módulo del coche . Luego accedemos a las clases que necesita a través de la sintaxis *module_name.ClassName .* En v creamos nuevamente un Volkswagen Beetle, y en w creamos un Tesla Roadster.

Importación de todas las clases de un módulo

Puede importar cada clase de un módulo usando la siguiente sintaxis:

```
de la importación de module_name *
```

Este método no se recomienda por dos razones. Primero, es útil poder leer las declaraciones de importación en la parte superior de un archivo y tener una idea clara de qué clases usa un programa. Con este enfoque, no está claro qué clases está utilizando del módulo. Este enfoque también puede generar confusión con los nombres en el archivo. Si importa accidentalmente una clase con el mismo nombre que otra cosa en su archivo de programa, puede crear errores que son difíciles de diagnosticar. Muestro esto aquí porque, aunque no es un enfoque recomendado, es probable que lo vea en el código de otras personas en algún momento.

Si necesita importar muchas clases de un módulo, es mejor importar el módulo completo y usar la sintaxis *module_name.ClassName .*
No verá todas las clases utilizadas en la parte superior del archivo, pero verá claramente dónde se utiliza el módulo en el programa. También evitará los posibles conflictos de nombres que pueden surgir cuando importa cada clase en un módulo.

Importación de un módulo en un módulo

A veces querrá distribuir sus clases en varios módulos para evitar que un archivo crezca demasiado y evitar almacenar clases no relacionadas en el mismo módulo. Cuando almacena sus clases en varios módulos, puede encontrar que una clase en un módulo depende de una clase en otro módulo. Cuando esto sucede, puede importar la clase requerida al primer módulo.

Por ejemplo, almacenemos la clase Car en un módulo y ElectricCar y clases de batería en un módulo separado. Haremos un nuevo módulo llamado *electric_car.py, reemplazando* el archivo *electric_car.py* que creamos anteriormente, y copiaremos solo las clases Battery y ElectricCar en este archivo:

coche_electrico.py

```
"""Un conjunto de clases que se pueden utilizar para representar coches eléctricos."""

u de importación de automóviles

batería de clase:
    --recorte--

clase ElectricCar(Coche):
    --recorte--
```

La clase ElectricCar necesita acceso a su clase principal Car, por lo que importamos Car directamente al módulo en u. Si olvidamos esta línea, Python generará un error cuando intentemos importar el módulo electric_car . También necesitamos actualizar el módulo Car para que contenga solo la clase Car :

coche.py

```
"""Una clase que se puede usar para representar un automóvil"."""

Coche de clase:

    --recorte--
```

Ahora podemos importar desde cada módulo por separado y crear cualquier tipo de automóvil que necesitemos:

```
my_cars.py u from car import Car
de electric_car importar ElectricCar

mi_escarabajo = Coche('volkswagen', 'escarabajo', 2019)
imprimir(mi_escarabajo.obtener_nombre_descriptivo())

my_tesla = ElectricCar('tesla', 'roadster', 2019)
imprimir (my_tesla.get_descriptive_name())
```

En u importamos Car desde su módulo y ElectricCar desde su módulo. Luego creamos un automóvil normal y un automóvil eléctrico. Ambos tipos de coches se crean correctamente:

```
Volkswagen Escarabajo 2019
Descapotable Tesla 2019
```

Uso de alias

Como vio en el Capítulo 8, los alias pueden ser muy útiles al usar módulos para organizar el código de sus proyectos. También puede usar alias al importar clases.

Como ejemplo, considere un programa en el que desea hacer un montón de autos eléctricos. Puede volverse tedioso escribir (y leer) ElectricCar una y otra vez. Puede dar a ElectricCar un alias en la declaración de importación:

```
de electric_car importar ElectricCar como CE
```

Ahora puedes usar este alias cuando quieras hacer un auto eléctrico:

```
mi_tesla = EC('tesla', 'roadster', 2019)
```

Encontrar su propio flujo de trabajo

Como puede ver, Python le brinda muchas opciones sobre cómo estructurar el código en un proyecto grande. Es importante conocer todas estas posibilidades para que pueda determinar las mejores formas de organizar sus proyectos, así como comprender los proyectos de otras personas.

Cuando esté comenzando, mantenga la estructura de su código simple. Intente hacer todo en un solo archivo y mueva sus clases a módulos separados una vez que todo esté funcionando. Si le gusta cómo interactúan los módulos y los archivos, intente almacenar sus clases en módulos cuando inicie un proyecto. Encuentre un enfoque que le permita escribir código que funcione y continúe desde allí.

Inténtalo tú mismo

9-10. Restaurante importado: utilizando su última clase de restaurante , guárdela en un módulo. Cree un archivo separado que importe Restaurant. Cree una instancia de Restaurant y llame a uno de los métodos de Restaurant para mostrar que la declaración de importación funciona correctamente.

9-11. Administrador importado: Comience con su trabajo del Ejercicio 9-8 (página 173). Almacene las clases Usuario, Privilegios y Administrador en un módulo. Cree un archivo separado, cree una instancia de administrador y llame a show_privileges() para mostrar que todo funciona correctamente.

9-12. Múltiples módulos: almacene la clase de usuario en un módulo y almacene las clases de privilegios y administración en un módulo separado. En un archivo separado, cree una instancia de administrador y llame a show_privileges() para mostrar que todo sigue funcionando correctamente.

La biblioteca estándar de Python

La *biblioteca estándar de Python* es un conjunto de módulos incluidos con cada instalación de Python. Ahora que tiene una comprensión básica de cómo funcionan las funciones y las clases, puede comenzar a usar módulos como estos que han escrito otros programadores. Puede usar cualquier función o clase en la biblioteca estándar al incluir una declaración de importación simple en la parte superior de su archivo. Veamos un módulo, aleatorio, que puede ser útil para modelar muchas situaciones del mundo real.

Una función interesante del módulo aleatorio es randint(). Esta función toma dos argumentos enteros y devuelve un número entero seleccionado al azar entre (e incluyendo) esos números.

Aquí se explica cómo generar un número aleatorio entre 1 y 6:

```
>>> de randint de importación aleatoria
>>> citas (1, 6)
3
```

Otra función útil es choice(). Esta función toma una lista o tupla y devuelve un elemento elegido al azar:

```
>>> de la elección de importación aleatoria
>>> jugadores = ['charles', 'martina', 'michael', 'florencia', 'eli']
>>> first_up = elección(jugadores)
>>> primero_arriba
'florencia'
```

El módulo aleatorio no debe usarse al crear aplicaciones relacionadas con la seguridad, pero es lo suficientemente bueno para muchos proyectos divertidos e interesantes.

Nota *También puede descargar módulos de fuentes externas. Verá varios de estos ejemplos en la Parte II, donde necesitaremos módulos externos para completar cada proyecto.*

Inténtalo tú mismo

9-13. Dado: crea una clase de dado con un atributo llamado lados, que tiene un valor predeterminado de 6. Escribe un método llamado roll_die() que imprima un número aleatorio entre 1 y el número de lados que tiene el dado. Haz un dado de 6 caras y tíralo 10 veces.

Haz un dado de 10 caras y un dado de 20 caras. Tira cada dado 10 veces.

9-14. Lotería: Haz una lista o tupla que contenga una serie de 10 números y cinco letras. Seleccione al azar cuatro números o letras de la lista e imprima un mensaje diciendo que cualquier boleto que coincida con estos cuatro números o letras gana un premio.

9-15. Análisis de lotería: puede usar un ciclo para ver qué tan difícil puede ser ganar el tipo de lotería que acaba de modelar. Haz una lista o tupla llamada my_ticket. Escribe un bucle que siga sacando números hasta que gane tu boleto. Imprima un mensaje informando cuántas veces tuvo que ejecutarse el ciclo para obtener un boleto ganador.

9-16. Python Module of the Week: un excelente recurso para explorar la biblioteca estándar de Python es un sitio llamado Python Module of the Week. Vaya a https:// pymotw.com/ y mire la tabla de contenido. Encuentre un módulo que le parezca interesante y lea sobre él, tal vez comenzando con el aleatorio módulo.

Clases de estilismo

Vale la pena aclarar algunos problemas de estilo relacionados con las clases, especialmente a medida que sus programas se vuelven más complicados.

Los nombres de las clases deben escribirse en *CamelCase*. Para hacer esto, escriba en mayúscula la primera letra de cada palabra en el nombre y no use guiones bajos. Los nombres de instancias y módulos deben escribirse en minúsculas con guiones bajos entre palabras.

Cada clase debe tener una cadena de documentación inmediatamente después de la definición de la clase. nición La cadena de documentación debe ser una breve descripción de lo que hace la clase, y debe seguir las mismas convenciones de formato que usó para escribir cadenas de documentación en funciones. Cada módulo también debe tener una cadena de documentación que describa para qué se pueden usar las clases en un módulo.

Puede usar líneas en blanco para organizar el código, pero no las use en exceso. Dentro de una clase puede usar una línea en blanco entre los métodos y dentro de un módulo puede usar dos líneas en blanco para separar las clases.

Si necesita importar un módulo de la biblioteca estándar y un módulo que escribió, primero coloque la declaración de importación para el módulo de la biblioteca estándar. Luego agregue una línea en blanco y la declaración de importación para el módulo que escribió. En programas con múltiples declaraciones de importación, esta convención facilita ver de dónde provienen los diferentes módulos utilizados en el programa.

Resumen

En este capítulo aprendió a escribir sus propias clases. Aprendió cómo almacenar información en una clase usando atributos y cómo escribir métodos que le den a sus clases el comportamiento que necesitan. Aprendió a escribir métodos __init__() que crean instancias de sus clases con exactamente los atributos que desea. Viste cómo modificar los atributos de una instancia directamente ya través de métodos. Aprendió que la herencia puede simplificar la creación de clases que están relacionadas entre sí y aprendió a usar instancias de una clase como atributos en otra clase para mantener cada clase simple.

Viste cómo almacenar clases en módulos e importar clases necesitas en los archivos donde se usarán pueden mantener sus proyectos organizados. Comenzaste a aprender sobre la biblioteca estándar de Python y viste un ejemplo basado en el módulo aleatorio . Finalmente, aprendió a diseñar sus clases usando las convenciones de Python.

En el Capítulo 10, aprenderá a trabajar con archivos para que pueda guardar el trabajo que ha realizado en un programa y el trabajo que ha permitido que realicen los usuarios. También aprenderá sobre las *excepciones,* una clase especial de Python diseñada para ayudarlo a responder a los errores cuando surjan.

10

Ficheros y excepciones

Ahora que domina las habilidades básicas que necesita para escribir programas organizados que sean fáciles de usar, es hora de pensar en hacer que sus programas sean aún más relevantes y utilizables. En este capítulo aprenderá a trabajar con archivos para que sus programas puedan analizar rápidamente muchos datos.

Aprenderá a manejar los errores para que sus programas no se bloqueen cuando se encuentren con situaciones inesperadas. Aprenderá sobre las *excepciones,* que son objetos especiales que Python crea para administrar los errores que surgen mientras se ejecuta un programa. También aprenderá sobre el módulo json , que le permite guardar datos de usuario para que no se pierdan cuando su programa deja de ejecutarse.

Aprender a trabajar con archivos y guardar datos hará que sus programas sean más fáciles de usar para las personas. Los usuarios podrán elegir qué datos ingresar y cuándo ingresarlos. Las personas pueden ejecutar su programa, hacer algo de trabajo y luego cerrar el programa y continuar donde lo dejaron más tarde. Aprender a manejar excepciones lo ayudará a lidiar con situaciones en las que los archivos no existen y con otros problemas que pueden hacer que sus programas se bloqueen. Esto hará que sus programas sean más robustos cuando encuentren datos incorrectos, ya sea

proviene de errores inocentes o de intentos maliciosos de romper sus programas. Con las habilidades que aprenderá en este capítulo, hará que sus programas sean más aplicables, utilizables y estables.

Lectura de un archivo

Una increíble cantidad de datos está disponible en archivos de texto. Los archivos de texto pueden contener datos meteorológicos, datos de tráfico, datos socioeconómicos, obras literarias y más. Leer desde un archivo es particularmente útil en aplicaciones de análisis de datos, pero también es aplicable a cualquier situación en la que desee analizar o modificar información almacenada en un archivo. Por ejemplo, puede escribir un programa que lea el contenido de un archivo de texto y reescriba el archivo con un formato que permita que un navegador lo muestre.

Cuando desee trabajar con la información de un archivo de texto, el primer paso es leer el archivo en la memoria. Puede leer todo el contenido de un archivo o puede trabajar con el archivo una línea a la vez.

Lectura de un archivo completo

Para comenzar, necesitamos un archivo con unas pocas líneas de texto. Comencemos con un archivo que contiene *pi* con 30 decimales, con 10 decimales por línea:

pi_digitos.txt

```
3.1415926535
  8979323846
  2643383279
```

Para probar los siguientes ejemplos usted mismo, puede ingresar estas líneas en un editor y guardar el archivo como *pi_digits.txt,* o puede descargar el archivo de los recursos del libro a través de *https:// nostarch.com/ pythoncrashcourse2e/.* Guarde el archivo en el mismo directorio donde almacenará los programas de este capítulo.

Aquí hay un programa que abre este archivo, lo lee e imprime el contenido del archivo a la pantalla:

lector_de_archivos.py

```
con open('pi_digits.txt') como file_object:
    contenidos = objeto_archivo.leer()
imprimir (contenido)
```

La primera línea de este programa tiene mucho que hacer. Empecemos mirando la función open() . Para realizar cualquier trabajo con un archivo, incluso solo para imprimir su contenido, primero debe *abrir* el archivo para acceder a él. La función open() necesita un argumento: el nombre del archivo que desea abrir. Python busca este archivo en el directorio donde está almacenado el programa que se está ejecutando actualmente. En este ejemplo, *file_reader.py* se está ejecutando actualmente, por lo que Python busca *pi_digits.txt* en el directorio donde está almacenado *file_reader.py .* el abierto() La función devuelve un objeto que representa el archivo. Aquí, abra ('pi_digits.txt') devuelve un objeto que representa *pi_digits.txt.* Python asigna este objeto a file_object, con el que trabajaremos más adelante en el programa.

La palabra clave con cierra el archivo una vez que ya no es necesario acceder a él. Observe cómo llamamos a open() en este programa pero no a close(). Puede abrir y cerrar el archivo llamando a open() y close(), pero si un error en su programa impide que se ejecute el método close() , es posible que el archivo nunca se cierre. Esto puede parecer trivial, pero los archivos mal cerrados pueden hacer que los datos se pierdan o se dañen. Y si llama a close() demasiado pronto en su programa, se encontrará tratando de trabajar con un archivo *cerrado* (un archivo al que no puede acceder), lo que genera más errores. No siempre es fácil saber exactamente cuándo debe cerrar un archivo, pero con la estructura que se muestra aquí, Python lo resolverá por usted. Todo lo que tiene que hacer es abrir el archivo y trabajar con él como desee, confiando en que Python lo cerrará automáticamente cuando el bloque with termine de ejecutarse.

Una vez que tenemos un objeto de archivo que representa *pi_digits.txt,* usamos read() en la segunda línea de nuestro programa para leer todo el contenido del archivo y almacenarlo como una cadena larga en contenidos. Cuando imprimimos el valor de los contenidos, recuperamos el archivo de texto completo:

```
3.1415926535
  8979323846
  2643383279
```

La única diferencia entre esta salida y el archivo original es la línea en blanco adicional al final de la salida. La línea en blanco aparece porque read() devuelve una cadena vacía cuando llega al final del archivo; esta cadena vacía se muestra como una línea en blanco. Si desea eliminar la línea en blanco adicional, puede usar rstrip() en la llamada a print():

```
con open('pi_digits.txt') como file_object:
    contenidos = objeto_archivo.leer()
    imprimir (contenido.rstrip())
```

Recuerde que el método rstrip() de Python elimina, o elimina, cualquier carácter de espacio en blanco del lado derecho de una cadena. Ahora la salida coincide exactamente con el contenido del archivo original:

```
3.1415926535
  8979323846
  2643383279
```

Rutas de archivos

Cuando pasa un nombre de archivo simple como *pi_digits.txt* a la función open() , Python busca en el directorio donde está almacenado el archivo que se está ejecutando actualmente (es decir, su archivo de programa .py).

A veces, dependiendo de cómo organice su trabajo, el archivo que desea abrir no estará en el mismo directorio que su archivo de programa.
Por ejemplo, puede almacenar sus archivos de programa en una carpeta llamada

python_trabajo; dentro *de python_work,* es posible que tenga otra carpeta llamada *text_files* para distinguir sus archivos de programa de los archivos de texto que están manipulando. Aunque *text_files* está en *python_work,* solo pasa open() el nombre de un archivo en *text_files* no funcionará, porque Python solo buscará en *python_work* y se detendrá allí; no continuará y buscará en *text_files.* Para que Python abra archivos desde un directorio que no sea aquel donde está almacenado el archivo de su programa, debe proporcionar una *ruta de archivo,* que le indica a Python que busque en una ubicación específica de su sistema.

Debido a que *text_files* está dentro *de python_work,* puede usar una ruta de archivo relativa para abrir un archivo desde *text_files.* Una *ruta de archivo relativa* le dice a Python que busque una ubicación determinada en relación con el directorio donde se almacena el archivo del programa que se está ejecutando actualmente. Por ejemplo, escribirías:

```
con open('text_files/filename.txt') como file_object:
```

Esta línea le dice a Python que busque el archivo .txt deseado en la carpeta *text_files* y asume que *text_files* se encuentra dentro *de python_work* (que es).

sistemas Windows usan una barra invertida (\) en lugar de una barra inclinada (/) al mostrar las rutas de los archivos, pero aún puede usar barras inclinadas en su código.

También puede decirle a Python exactamente dónde está el archivo en su computadora, independientemente de dónde esté almacenado el programa que se está ejecutando. Esto se llama una *ruta de archivo absoluta.* Utiliza una ruta absoluta si una ruta relativa no funciona. Por ejemplo, si ha colocado *archivos* de texto en alguna carpeta que no sea *python_work, digamos,* una carpeta llamada *otros archivos,* simplemente pasando open() la ruta *'text_files/ filename.txt'* no funcionará porque Python solo buscará esa ubicación. dentro *de python_work.* Deberá escribir una ruta completa para aclarar dónde desea que busque Python.

Las rutas absolutas suelen ser más largas que las rutas relativas, por lo que es útil asígnelos a una variable y luego pase esa variable a open():

```
file_path = '/home/ehmatthes/other_files/text_files/filename.txt'
con open(file_path) como file_object:
```

Usando rutas absolutas, puede leer archivos desde cualquier ubicación en su sistema. Por ahora es más fácil almacenar archivos en el mismo directorio que sus archivos de programa o en una carpeta como *text_files* dentro del directorio que almacena sus archivos de programa.

Si intenta usar barras invertidas en una ruta de archivo, obtendrá un error porque la barra invertida se usa para escapar de los caracteres en las cadenas. Por ejemplo, en la ruta "C:\ruta\a\archivo.txt", la secuencia \t se interpreta como una tabulación. Si necesita usar barras invertidas, puede escapar de cada una en la ruta, así: "C:\\ruta\\a\\archivo.txt".

Lectura línea por línea

Cuando esté leyendo un archivo, a menudo querrá examinar cada línea del archivo. Es posible que esté buscando cierta información en el archivo o que desee modificar el texto del archivo de alguna manera. Por ejemplo, es posible que desee leer un archivo de datos meteorológicos y trabajar con cualquier línea que incluya la palabra *soleado* en la descripción del tiempo de ese día. En un informe de noticias, puede buscar cualquier línea con la etiqueta <titular> y volver a escribir esa línea con un tipo específico de formato.

Puede usar un bucle for en el objeto de archivo para examinar cada línea de un archivo de una en una:

```
file_reader.py u nombre de archivo = 'pi_digits.txt'

v con open(nombre de archivo) como objeto_archivo:
w para línea en file_object:
        imprimir (línea)
```

En u asignamos el nombre del archivo que estamos leyendo a la variable nombre del archivo. Esta es una convención común cuando se trabaja con archivos. Debido a que el nombre de archivo variable no representa el archivo real, es solo una cadena que le dice a Python dónde encontrar el archivo, puede intercambiar fácilmente 'pi_digits.txt' para el nombre de otro archivo con el que desea trabajar. Después de llamar a open(), se asigna un objeto que representa el archivo y su contenido a la variable file_object v. Nuevamente usamos la sintaxis with para permitir que Python abra y cierre el archivo correctamente. Para examinar el contenido del archivo, trabajamos en cada línea del archivo haciendo un bucle sobre el objeto de archivo w.

Cuando imprimimos cada línea, encontramos aún más líneas en blanco:

```
3.1415926535

  8979323846

  2643383279
```

Estas líneas en blanco aparecen porque hay un carácter invisible de nueva línea al final de cada línea en el archivo de texto. La función de impresión agrega su propia nueva línea cada vez que la llamamos, por lo que terminamos con dos caracteres de nueva línea al final de cada línea: uno del archivo y otro de print(). Usando rstrip() en cada línea de la llamada print() elimina estas líneas en blanco adicionales:

```
nombre de archivo = 'pi_digits.txt'

con abierto (nombre de archivo) como file_object:
    para línea en file_object:
        imprimir (línea.rstrip())
```

Ahora la salida coincide con el contenido del archivo una vez más:

```
3.1415926535
  8979323846
  2643383279
```

Hacer una lista de líneas de un archivo

Cuando usa with, el objeto de archivo devuelto por open() solo está disponible dentro del bloque with que lo contiene. Si desea conservar el acceso al contenido de un archivo fuera del bloque with , puede almacenar las líneas del archivo en una lista dentro del bloque y luego trabajar con esa lista. Puede procesar partes del archivo inmediatamente y posponer parte del procesamiento para más adelante en el programa.

El siguiente ejemplo almacena las líneas de *pi_digits.txt* en una lista dentro del bloque with y luego imprime las líneas fuera del bloque with :

```
nombre de archivo = 'pi_digits.txt'

con abierto (nombre de archivo) como file_object:
u líneas = file_object.readlines()

v para línea en líneas:
    imprimir (línea.rstrip())
```

En u, el método readlines() toma cada línea del archivo y la almacena en una lista. Luego, esta lista se asigna a las líneas, con las que podemos continuar trabajando después de que finalice el bloque with . En v usamos un bucle for simple para imprimir cada línea de líneas. Dado que cada elemento de las líneas corresponde a cada línea del archivo, la salida coincide exactamente con el contenido del archivo.

Trabajar con el contenido de un archivo

Después de leer un archivo en la memoria, puede hacer lo que quiera con esos datos, así que exploremos brevemente los dígitos de *pi.* Primero, intentaremos construir una sola cadena que contenga todos los dígitos del archivo sin espacios en blanco:

pi_string.py

```
nombre de archivo = 'pi_digits.txt'

con abierto (nombre de archivo) como file_object:
    líneas = objeto_archivo.readlines()

u pi_string = ''
v para línea en líneas:
    pi_string += línea.rstrip()

w imprimir (pi_cadena)
  imprimir (len (pi_cadena))
```

Comenzamos abriendo el archivo y almacenando cada línea de dígitos en una lista, solo como hicimos en el ejemplo anterior. En u creamos una variable, pi_string, para contener los dígitos de *pi.* Luego creamos un ciclo que agrega cada línea de dígitos a pi_string y elimina el carácter de nueva línea de cada línea v. En w imprimimos esta cadena y también mostramos su longitud:

```
3.1415926535 8979323846 2643383279 36
```

La variable pi_string contiene el espacio en blanco que estaba en el lado izquierdo de los dígitos en cada línea, pero podemos deshacernos de eso usando strip() en lugar de rstrip():

```
--recorte--
para línea en líneas:
    pi_string += línea.strip()

imprimir (pi_cadena)
imprimir (len (pi_cadena))
```

Ahora tenemos una cadena que contiene *pi* con 30 decimales. La cadena tiene 32 caracteres porque también incluye el 3 inicial y un punto decimal:

```
3.141592653589793238462643383279
32
```

Cuando Python lee un archivo de texto, interpreta todo el texto del archivo como una cadena. Si lee un número y quiere trabajar con ese valor en un contexto numérico, tendrá que convertirlo en un número entero usando la función *int() o convertirlo en un número flotante usando la* función *float() .*

Archivos grandes: un millón de dígitos

Hasta ahora nos hemos centrado en analizar un archivo de texto que contiene solo tres líneas, pero el código de estos ejemplos funcionaría igual de bien en archivos mucho más grandes. Si comenzamos con un archivo de texto que contiene *pi* hasta 1.000.000 de decimales en lugar de solo 30, podemos crear una sola cadena que contenga todos estos dígitos. No necesitamos cambiar nuestro programa en absoluto excepto para pasarle un archivo diferente. También imprimiremos solo los primeros 50 lugares decimales, para que no tengamos que ver pasar un millón de dígitos en la terminal:

pi_string.py

```
nombre de archivo = 'pi_million_digits.txt'

con abierto (nombre de archivo) como file_object:
    líneas = objeto_archivo.readlines()
```

```
pi_string = ''
para línea en líneas:
    pi_string += línea.strip()

imprimir(f"{pi_cadena[:52]}...")
imprimir (len (pi_cadena))
```

El resultado muestra que, de hecho, tenemos una cadena que contiene *pi* con 1,000,000 de decimales:

```
3.14159265358979323846264338327950288419716939937510...
1000002
```

Python no tiene un límite inherente a la cantidad de datos con los que puede trabajar; usted puede trabajar con tantos datos como la memoria de su sistema pueda manejar.

Para ejecutar este programa (y muchos de los ejemplos que siguen), deberá descargar los recursos disponibles en https://nostarch.com/pythoncrashcourse2e/.

¿Tu cumpleaños está contenido en Pi?

Siempre he tenido curiosidad por saber si mi cumpleaños aparece en algún lugar de los dígitos de *pi.* Usemos el programa que acabamos de escribir para averiguar si el cumpleaños de alguien aparece en algún lugar del primer millón de dígitos de *pi.* Podemos hacer esto expresando cada cumpleaños como una cadena de dígitos y viendo si esa cadena aparece en algún lugar de pi_string:

```
--recorte--
para línea en líneas:
    pi_string += línea.strip()

u cumpleaños = entrada ("Ingrese su cumpleaños, en la forma mmddyy:")
v si cumpleaños en pi_string:
    print("¡Tu cumpleaños aparece en el primer millón de dígitos de pi!")
demás:
    print("Tu cumpleaños no aparece en el primer millón de dígitos de pi.")
```

En u solicitamos el cumpleaños del usuario, y luego en v verificamos si ese la cadena está en pi_string. Vamos a intentarlo:

```
Ingrese su fecha de nacimiento, en la forma mmddyy:
120372 ¡ Su cumpleaños aparece en el primer millón de dígitos de pi!
```

¡Mi cumpleaños aparece en los dígitos de *pi!* Una vez que hayas leído de un archivo, puede analizar su contenido en casi cualquier forma que pueda imaginar.

Inténtalo tú mismo

10-1. Aprendiendo Python: abra un archivo en blanco en su editor de texto y escriba unas pocas líneas que resuman lo que ha aprendido sobre Python hasta ahora. Comience cada línea con la frase In Python you can. . . . Guarde el archivo como learning_python.txt en el mismo directorio que sus ejercicios de este capítulo. Escriba un programa que lea el archivo e imprima lo que escribió tres veces. Imprima el contenido una vez leyendo todo el archivo, una vez recorriendo el objeto del archivo y una vez almacenando las líneas en una lista y luego trabajando con ellas fuera del bloque with .

10-2. Aprendiendo C: puede usar el método replace() para reemplazar cualquier palabra en una cadena con una palabra diferente. Aquí hay un ejemplo rápido que muestra cómo reemplazar 'perro' con 'gato' en una oración:

```
>>> mensaje = "Me gustan mucho los perros".
>>> mensaje.reemplazar('perro', 'gato')
Me gustan mucho los gatos.
```

Lea cada línea del archivo que acaba de crear, learning_python.txt, y reemplace la palabra Python con el nombre de otro idioma, como C. Imprima cada línea modificada en la pantalla.

Escribir en un archivo

Una de las formas más sencillas de guardar datos es escribirlos en un archivo. Cuando escribe texto en un archivo, la salida seguirá estando disponible después de cerrar la terminal que contiene la salida de su programa. Puede examinar la salida después de que un programa termine de ejecutarse y también puede compartir los archivos de salida con otros. También puede escribir programas que vuelvan a leer el texto en la memoria y vuelvan a trabajar con él más tarde.

Escribir en un archivo vacío

Para escribir texto en un archivo, debe llamar a open() con un segundo argumento que le dice a Python que desea escribir en el archivo. Para ver cómo funciona esto, escribamos un mensaje simple y almacenémoslo en un archivo en lugar de imprimirlo en la pantalla:

escribe _mensaje.py

```
nombre de archivo = 'programación.txt'

u con open(nombre de archivo, 'w') como objeto_archivo:
v file_object.write("Me encanta programar")
```

La llamada a open() en este ejemplo tiene dos argumentos u. El primer argumento sigue siendo el nombre del archivo que queremos abrir. El segundo argumento, 'w', le dice a Python que queremos abrir el archivo en *modo de escritura.* Puedes abrir un archivo

en *modo de lectura* ('r'), *modo de escritura* ('w'), *modo* de adición ('a') o un modo que le permita leer y escribir en el archivo ('r+'). Si omite el argumento de modo, Python abre el archivo en modo de solo lectura de forma predeterminada.

La función open() crea automáticamente el archivo en el que está escribiendo si aún no existe. Sin embargo, tenga cuidado al abrir un archivo en modo de escritura ('w') porque si el archivo existe, Python borrará el contenido del archivo antes de devolver el objeto del archivo.

En v usamos el método write() en el objeto de archivo para escribir una cadena en el archivo. Este programa no tiene salida de terminal, pero si abre el archivo *programación.txt,* verá una línea:

```
programación.txt Me encanta programar.
```

Este archivo se comporta como cualquier otro archivo en su computadora. Puede abrirlo, escribir texto nuevo en él, copiarlo, pegarlo, etc.

Python solo puede escribir cadenas en un archivo de texto. Si desea almacenar datos numéricos en un archivo de texto, primero tendrá que convertir los datos a formato de cadena usando la función *str() .*

Escribir varias líneas

La función write() no agrega líneas nuevas al texto que escribe. Por lo tanto, si escribe más de una línea sin incluir caracteres de nueva línea, es posible que su archivo no tenga el aspecto que desea:

```
nombre de archivo = 'programación.txt'

con open(nombre de archivo, 'w') como objeto_archivo:
    file_object.write("Me encanta programar")
    file_object.write("Me encanta crear nuevos juegos").
```

Si abres *programación.txt,* verás las dos líneas juntas:

```
Me encanta programar. Me encanta crear nuevos juegos.
```

Incluir líneas nuevas en tus llamadas a write() hace que cada cadena aparezca en su propia línea:

```
nombre de archivo = 'programación.txt'

con open(nombre de archivo, 'w') como objeto_archivo:
    file_object.write("Me encanta programar.\n")
    file_object.write("Me encanta crear nuevos juegos.\n")
```

La salida ahora aparece en líneas separadas:

```
Me encanta la programación.
Me encanta crear nuevos juegos.
```

También puede utilizar espacios, caracteres de tabulación y líneas en blanco para dar formato a su salida, tal como lo ha estado haciendo con la salida basada en terminal.

Agregar a un archivo

Si desea agregar contenido a un archivo en lugar de escribir sobre el contenido existente, puede abrir el archivo en *modo de adición.* Cuando abre un archivo en modo de adición, Python no borra el contenido del archivo antes de devolver el objeto de archivo.
Cualquier línea que escriba en el archivo se agregará al final del archivo. Si el archivo aún no existe, Python creará un archivo vacío para usted.

Modifiquemos *write_message.py* agregando algunas nuevas razones por las que nos encanta programar al archivo de *programación existente.txt:*

escribe _mensaje.py

```
nombre de archivo = 'programación.txt'

u con open(filename, 'a') como file_object:
v file_object.write("También me encanta encontrar significado en grandes conjuntos de datos.\n")
    file_object.write("Me encanta crear aplicaciones que puedan ejecutarse en un navegador.\n")
```

En u usamos el argumento 'a' para abrir el archivo y agregarlo en lugar de escribir sobre el archivo existente. En v escribimos dos líneas nuevas, que se agregan a *programación.txt:*

programación.txt

```
Me encanta programar.
Me encanta crear nuevos juegos.
También me encanta encontrar significado en grandes conjuntos de datos.
Me encanta crear aplicaciones que puedan ejecutarse en un navegador.
```

Terminamos con el contenido original del archivo, seguido del nuevo contenido que acabamos de agregar.

Inténtalo tú mismo

10-3. Invitado: escriba un programa que solicite al usuario su nombre. Cuando respondan, escribe su nombre en un archivo llamado guest.txt.

10-4. Libro de invitados: escriba un bucle while que solicite a los usuarios su nombre. Cuando ingresen su nombre, imprima un saludo en la pantalla y agregue una línea que registre su visita en un archivo llamado guest_book.txt. Asegúrese de que cada entrada aparezca en una nueva línea en el archivo.

10-5. Encuesta de programación: escriba un ciclo while que pregunte a las personas por qué les gusta programar. Cada vez que alguien ingrese un motivo, agregue su motivo a un archivo que almacena todas las respuestas.

Excepciones

Python usa objetos especiales llamados *excepciones* para administrar los errores que surgen durante la ejecución de un programa. Cada vez que ocurre un error que hace que Python no esté seguro de qué hacer a continuación, crea un objeto de excepción. Si escribe código que maneja la excepción, el programa continuará ejecutándose. Si no maneja la excepción, el programa se detendrá y mostrará un *seguimiento,* que incluye un informe de la excepción que se generó.

Las excepciones se manejan con bloques try-except . Un bloque try-except le pide a Python que haga algo, pero también le dice a Python qué hacer si surge una excepción. Cuando usa bloques try-except , sus programas continuarán ejecutándose incluso si las cosas comienzan a salir mal. En lugar de rastreos, que pueden resultar confusos para los usuarios, los usuarios verán mensajes de error amigables que usted escribe.

Manejo de la excepción ZeroDivisionError

Veamos un error simple que hace que Python genere una excepción. Probablemente sepa que es imposible dividir un número por cero, pero pidamos a Python que lo haga de todos modos:

división
_calculadora.py

```
imprimir (5/0)
```

Por supuesto, Python no puede hacer esto, por lo que obtenemos un seguimiento:

```
Rastreo (llamadas recientes más última):
  Archivo "division_calculator.py", línea 1, en <módulo> print(5/0) u
    ZeroDivisionError: división por cero
```

El error informado en u en el rastreo, ZeroDivisionError, es una excepción . objeto de ción. Python crea este tipo de objeto en respuesta a una situación en la que no puede hacer lo que le pedimos. Cuando esto sucede, Python detiene el programa y nos dice el tipo de excepción que se generó. Podemos utilizar esta información para modificar nuestro programa. Le diremos a Python qué hacer cuando ocurra este tipo de excepción; así, si vuelve a pasar, estamos preparados.

Uso de bloques try-except

Cuando crea que puede ocurrir un error, puede escribir un bloque try-except para manejar la excepción que podría generarse. Le dices a Python que intente ejecutar algún código y le dices qué hacer si el código resulta en un tipo particular de excepción.

Así es como se ve un bloque try-except para manejar la excepción ZeroDivisionError :

```
tratar:
    imprimir (5/0)
excepto ZeroDivisionError:
    print("¡No puedes dividir por cero!")
```

Ponemos print(5/0), la línea que causó el error, dentro de un bloque de prueba . Si el código en un bloque de prueba funciona, Python salta el bloque de excepción . Si el código en el bloque de prueba genera un error, Python busca un bloque de excepción cuyo error coincida con el que se generó y ejecuta el código en ese bloque.

En este ejemplo, el código en el bloque de prueba produce un ZeroDivisionError, por lo que Python busca un bloque de excepción que le indique cómo responder. Luego, Python ejecuta el código en ese bloque y el usuario ve un mensaje de error amigable en lugar de un rastreo:

```
¡No puedes dividir por cero!
```

Si hubiera más código seguido del bloque try-except , el programa continuaría ejecutándose porque le dijimos a Python cómo manejar el error. Veamos un ejemplo en el que detectar un error puede permitir que un programa continúe ejecutándose.

Uso de excepciones para evitar bloqueos

El manejo correcto de los errores es especialmente importante cuando el programa tiene más trabajo que hacer después de que ocurre el error. Esto sucede a menudo en programas que solicitan a los usuarios que ingresen información. Si el programa responde adecuadamente a una entrada no válida, puede solicitar una entrada más válida en lugar de bloquearse.

Vamos a crear una calculadora simple que solo haga divisiones:

división _calculadora.py

```
print("Dame dos numeros y los dividire")
print("Ingrese 'q' para salir.")

mientras que es cierto:
u primer_número = entrada("\nPrimer número: ")
        si primer_número == 'q':
            romper
v segundo_numero = entrada("Segundo numero: ")
        si segundo_número == 'q':
            romper
w respuesta = int(primer_número) / int(segundo_número)
        imprimir (respuesta)
```

Este programa solicita al usuario que ingrese un primer_número u y, si el usuario no ingresa *q* para salir, un segundo_número v. Luego dividimos estos dos números para obtener una respuesta w. Este programa no hace nada para manejar los errores, por lo que pedirle que divida por cero hace que se bloquee:

```
Dame dos números y los dividiré.
Introduzca 'q' para salir.

Primer número: 5
Segundo número: 0
Rastreo (llamadas recientes más última):
  Archivo "division_calculator.py", línea 9, en <módulo> respuesta =
    int(primer_número) / int(segundo_número)
ZeroDivisionError: división por cero
```

Es malo que el programa se bloquee, pero tampoco es una buena idea permitir que los usuarios vean los rastreos. Los usuarios no técnicos se sentirán confundidos por ellos y, en un entorno malicioso, los atacantes aprenderán más de lo que usted quiere que sepan a partir de un rastreo. Por ejemplo, sabrán el nombre de su archivo de programa y verán una parte de su código que no funciona correctamente. Un atacante habilidoso a veces puede usar esta información para determinar qué tipo de ataques usar contra su código.

El otro bloque

Podemos hacer que este programa sea más resistente a errores envolviendo la línea que podría producir errores en un bloque try-except . El error se produce en la línea que realiza la división, por lo que allí colocaremos el bloque try-except . Este ejemplo también incluye un bloque else . Cualquier código que dependa del intento . la ejecución exitosa del bloque va en el bloque else :

```
--recorte--
mientras que es cierto:
    --recorte--
    si segundo_número == 'q':
        romper
Inténtalo tu:
        respuesta = int(primer_número) / int(segundo_número)
v excepto ZeroDivisionError:
        print("¡No puedes dividir por 0!")
en otro:
        imprimir (respuesta)
```

Le pedimos a Python que intente completar la operación de división en un intento . bloque u, que incluye solo el código que podría causar un error. Cualquier código que dependa del éxito del bloque try se agrega al bloque else . En este caso, si la operación de división es exitosa, usamos el bloque else para imprimir el resultado w.

El bloque de excepción le dice a Python cómo responder cuando un ZeroDivisionError surge v. Si el bloque try no tiene éxito debido a un error de división por cero, imprimimos un mensaje amigable que le dice al usuario cómo evitar este tipo de error. El programa continúa ejecutándose y el usuario nunca ve un rastro:

```
Dame dos números y los dividiré.
Introduzca 'q' para salir.

Primer número: 5
Segundo número: 0
¡No puedes dividir por 0!
```

```
Primer número: 5
Segundo número: 2
2.5

primer número: q
```

El bloque try-except-else funciona así: Python intenta ejecutar el código en el bloque try . El único código que debe ir en un bloque de prueba es el código que puede provocar que se genere una excepción. A veces tendrá código adicional que debería ejecutarse solo si el bloque de prueba fue exitoso; este código va en el bloque else . El bloque de excepción le dice a Python qué hacer en caso de que surja una determinada excepción cuando intente ejecutar el código en el bloque de prueba .

Al anticipar las posibles fuentes de errores, puede escribir programas robustos que continúan ejecutándose incluso cuando encuentran datos no válidos y faltan recursos. Su código será resistente a errores de usuarios inocentes y ataques maliciosos.

Manejo de la excepción FileNotFoundError

Un problema común cuando se trabaja con archivos es el manejo de archivos que faltan. El archivo que está buscando puede estar en una ubicación diferente, el nombre del archivo puede estar mal escrito o el archivo puede no existir en absoluto. Puede manejar todas estas situaciones de una manera sencilla con un bloque de prueba excepto .

Intentemos leer un archivo que no existe. El siguiente programa intenta leer el contenido de *Alicia en el país de las maravillas,* pero no he guardado el archivo *alice.txt* en el mismo directorio que *alice.py:*

alice.py

```
nombre de archivo = 'alicia.txt'

con abierto (nombre de archivo, codificación = 'utf-8') como f:
    contenido = f.leer()
```

Hay dos cambios aquí. Uno es el uso de la variable f para representar el objeto de archivo, que es una convención común. El segundo es el uso del argumento de codificación . Este argumento es necesario cuando la codificación predeterminada de su sistema no coincide con la codificación del archivo que se está leyendo.

Python no puede leer un archivo que falta, por lo que genera una excepción:

```
Rastreo (llamadas recientes más última):
  Archivo "alice.py", línea 3, en <módulo>
    con abierto (nombre de archivo, codificación = 'utf-8') como f:
FileNotFoundError: [Errno 2] No existe tal archivo o directorio: 'alice.txt'
```

La última línea del rastreo informa un FileNotFoundError: esta es la excepción que Python crea cuando no puede encontrar el archivo que está tratando de abrir.

En este ejemplo, la función open() produce el error, así que para manejarlo, el bloque try comenzará con la línea que contiene open():

```
nombre de archivo = 'alicia.txt'

tratar:
    con abierto (nombre de archivo, codificación = 'utf-8') como f:
        contenido = f.leer()
excepto FileNotFoundError:
    print(f"Lo siento, el archivo {nombre de archivo} no existe").
```

En este ejemplo, el código en el bloque try produce un FileNotFoundError, entonces Python busca un bloque de excepción que coincida con ese error. Luego, Python ejecuta el código en ese bloque y el resultado es un mensaje de error amigable en lugar de un rastreo:

```
Lo sentimos, el archivo alice.txt no existe.
```

El programa no tiene nada más que hacer si el archivo no existe, por lo que el código de manejo de errores no agrega mucho a este programa. Construyamos sobre este ejemplo y veamos cómo el manejo de excepciones puede ayudar cuando trabaja con más de un archivo.

Analizando texto

Puede analizar archivos de texto que contengan libros completos. Muchas obras clásicas de la literatura están disponibles como archivos de texto simples porque son de dominio público. Los textos utilizados en esta sección provienen del Proyecto Gutenberg *(http:// gutenberg .org/).* Project Gutenberg mantiene una colección de obras literarias que están disponibles en el dominio público y es un gran recurso si está interesado en trabajar con textos literarios en sus proyectos de programación.

Extraigamos el texto de *Alicia en el país de las maravillas* e intentemos contar el número de palabras del texto. Usaremos el método de cadena split(), que puede construir una lista de palabras a partir de una cadena. Esto es lo que hace split() con una cadena que contiene solo el título "Alicia en el país de las maravillas":

```
>>> título = "Alicia en el País de las
Maravillas" >>> título.split()
['Alicia en el país de las Maravillas']
```

El método split() separa una cadena en partes siempre que encuentra un espacio y almacena todas las partes de la cadena en una lista. El resultado es una lista de palabras de la cadena, aunque también pueden aparecer algunos signos de puntuación con algunas de las palabras. Para contar el número de palabras en *Alicia en el país de las maravillas,* usaremos split() en todo el texto. Luego contaremos los elementos de la lista para tener una idea aproximada de la cantidad de palabras en el texto:

```
nombre de archivo = 'alicia.txt'

tratar:
```

```
        con abierto (nombre de archivo, codificación = 'utf-8') como f:
            contenido = f.leer()
    excepto FileNotFoundError:
        print(f"Lo siento, el archivo {nombre de archivo} no existe").
    demás:
        # Cuente el número aproximado de palabras en el archivo.
u palabras = contenidos.split()
v num_palabras = len(palabras)
w print(f"El archivo {filename} tiene aproximadamente {num_words} palabras").
```

Moví el archivo *alice.txt* al directorio correcto, por lo que el bloque de prueba funcionará esta vez. En u, tomamos el contenido de la cadena, que ahora contiene el texto completo de *Alicia en el país de las maravillas* como una cadena larga, y usamos split() método para producir una lista de todas las palabras en el libro. Cuando usamos len() en esta lista para examinar su longitud, obtenemos una buena aproximación del número de palabras en la cadena original v. En w imprimimos una declaración que informa cuántas palabras se encontraron en el archivo. Este código se coloca en el bloque else porque funcionará solo si el código en el bloque try se ejecutó con éxito por completo. El resultado nos dice cuántas palabras hay en *alice.txt:*

```
El archivo alice.txt tiene alrededor de 29465 palabras.
```

El conteo es un poco alto porque la información adicional es provista por el editor en el archivo de texto utilizado aquí, pero es una buena aproximación de la longitud de *Alicia en el País de las Maravillas.*

Trabajar con varios archivos

Agreguemos más libros para analizar. Pero antes de hacerlo, pasemos la mayor parte de este programa a una función llamada contar_palabras(). Al hacerlo, será más fácil ejecutar el análisis de varios libros:

```
word_count.py def count_words(nombre de archivo):
u """Cuenta el número aproximado de palabras en un archivo."""
    tratar:
        con abierto (nombre de archivo, codificación = 'utf-8') como f:
            contenidos = f.read() excepto
    FileNotFoundError:
        print(f"Lo siento, el archivo {nombre de archivo} no existe").
    demás:
        palabras = contenido.split()
        num_palabras = len(palabras)
        print(f"El archivo {filename} tiene aproximadamente {num_words} palabras").

nombre de archivo = 'alicia.txt'
contar_palabras(nombre de archivo)
```

La mayor parte de este código no ha cambiado. Simplemente lo sangramos y lo movimos al cuerpo de count_words(). Es un buen hábito mantener los comentarios actualizados cuando está modificando un programa, así que cambiamos el comentario a una cadena de documentos y lo reformulamos ligeramente.

Ahora podemos escribir un ciclo simple para contar las palabras en cualquier texto que queramos analizar. Hacemos esto almacenando los nombres de los archivos que queremos analizar en una lista y luego llamamos a count_words() para cada archivo en la lista. Intentaremos contar las palabras de *Alicia en el país de las maravillas, Siddhartha, Moby Dick* y *Mujercitas,* que están disponibles en el dominio público. He dejado intencionalmente *siddhartha.txt* fuera del directorio que contiene *word_count.py,* para que podamos ver qué tan bien nuestro programa maneja un archivo faltante:

```
def contar_palabras(nombre de archivo):
    --recorte--

nombres de archivo = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt', 'little_women.txt']
para nombre de archivo en nombres de archivo:
    contar_palabras(nombre de archivo)
```

El archivo *siddhartha.txt* faltante no tiene efecto en el resto de la ejecución del programa:

```
El archivo alice.txt tiene alrededor de 29465 palabras.
Lo sentimos, el archivo siddhartha.txt no existe.
El archivo moby_dick.txt tiene alrededor de 215830 palabras.
El archivo little_women.txt tiene alrededor de 189079 palabras.
```

El uso del bloque try-except en este ejemplo proporciona dos ventajas significativas. Evitamos que nuestros usuarios vean un rastreo y dejamos que el programa continúe analizando los textos que puede encontrar. Si no detectamos el FileNotFoundError que generó *siddhartha.txt* , el usuario vería un rastreo completo y el programa dejaría de ejecutarse después de intentar analizar *Siddhartha.* Nunca analizaría *Moby Dick* o *Mujercitas.*

fallando en silencio

En el ejemplo anterior, informamos a nuestros usuarios que uno de los archivos no estaba disponible. Pero no necesita informar cada excepción que detecte.
A veces querrá que el programa falle silenciosamente cuando ocurra una excepción y continúe como si nada hubiera pasado. Para hacer que un programa falle silenciosamente, escribes un bloque de prueba como de costumbre, pero le dices explícitamente a Python que no haga nada en el bloque de excepción . Python tiene una declaración de paso que le dice que no haga nada en un bloque:

```
def contar_palabras(nombre de archivo):
    """Cuenta el número aproximado de palabras en un archivo."""
    tratar:
        --recorte--
    excepto FileNotFoundError:
en      aprobar
    demás:
        --recorte--

nombres de archivo = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt', 'little_women.txt']
para nombre de archivo en nombres de archivo:
    contar_palabras(nombre de archivo)
```

La única diferencia entre este listado y el anterior es la declaración de aprobación en u. Ahora, cuando se genera un FileNotFoundError , se ejecuta el código en el bloque excepto , pero no sucede nada. No se produce ningún rastreo y no hay ningún resultado en respuesta al error que se generó. Los usuarios ven los recuentos de palabras para cada archivo que existe, pero no ven ninguna indicación de que no se encontró un archivo:

```
El archivo alice.txt tiene alrededor de 29465 palabras.
El archivo moby_dick.txt tiene alrededor de 215830 palabras.
El archivo little_women.txt tiene alrededor de 189079 palabras.
```

La instrucción de paso también actúa como marcador de posición. Es un recordatorio de que eres elegir no hacer nada en un punto específico de la ejecución de su programa y que es posible que desee hacer algo allí más tarde. Por ejemplo, en este programa podríamos decidir escribir cualquier nombre de archivo que falte en un archivo llamado *archivos_faltantes.txt.* Nuestros usuarios no verían este archivo, pero podríamos leerlo y solucionar cualquier texto que falte.

Decidir qué errores informar

¿Cómo sabe cuándo informar un error a sus usuarios y cuándo fallar en silencio? Si los usuarios saben qué textos se supone que deben analizarse, podrían apreciar un mensaje que les informe por qué no se analizaron algunos textos. Si los usuarios esperan ver algunos resultados pero no saben qué libros deben analizarse, es posible que no necesiten saber que algunos textos no estaban disponibles. Dar a los usuarios información que no están buscando puede disminuir la usabilidad de su programa. Las estructuras de manejo de errores de Python le brindan un control detallado sobre cuánto compartir con los usuarios cuando las cosas salen mal; depende de usted decidir cuánta información compartir.

El código bien escrito y debidamente probado no es muy propenso a errores internos, como errores de sintaxis o lógicos. Pero cada vez que su programa depende de algo externo, como la entrada del usuario, la existencia de un archivo o la disponibilidad de una conexión de red, existe la posibilidad de que se genere una excepción. Un poco de experiencia lo ayudará a saber dónde incluir bloques de manejo de excepciones en su programa y cuánto informar a los usuarios sobre los errores que surjan.

Inténtalo tú mismo

10-6. Adición: un problema común cuando se solicita una entrada numérica ocurre cuando las personas proporcionan texto en lugar de números. Cuando intente convertir la entrada a un int, obtendrá un ValueError. Escriba un programa que solicite dos números. Súmalos e imprime el resultado. Detecte el ValueError si alguno de los valores de entrada no es un número e imprima un mensaje de error amistoso. Pruebe su programa ingresando dos números y luego ingresando algún texto en lugar de un número.

(continuado)

10-7. Calculadora de sumas: Envuelva su código del Ejercicio 10-6 en un ciclo while para que el usuario pueda continuar ingresando números incluso si comete un error e ingresa texto en lugar de un número.

10-8. Perros y gatos: haga dos archivos, cats.txt y dogs.txt. Guarde al menos tres nombres de gatos en el primer archivo y tres nombres de perros en el segundo archivo. Escriba un programa que intente leer estos archivos e imprima el contenido del archivo en la pantalla. Envuelva su código en un bloque try-except para detectar el error FileNotFound e imprima un mensaje amigable si falta un archivo. Mueva uno de los archivos a una ubicación diferente en su sistema y asegúrese de que el código en el bloque excepto se ejecute correctamente.

10-9. Gatos y perros silenciosos: modifique su bloque de excepción en el ejercicio 10-8 para fallar silenciosamente si falta algún archivo.

10-10. Palabras comunes: Visite Project Gutenberg (https://gutenberg.org/) y encuentre algunos textos que le gustaría analizar. Descargue los archivos de texto de estos trabajos o copie el texto sin formato de su navegador a un archivo de texto en su computadora.

Puede usar el método count() para averiguar cuántas veces aparece una palabra o frase en una cadena. Por ejemplo, el siguiente código cuenta el número de veces que aparece 'fila' en una cadena:

```
>>> linea = "Rema, rema, rema tu bote" >>>
linea.count('fila')
2
>>> linea.inferior().cuenta('fila')
3
```

Tenga en cuenta que convertir la cadena a minúsculas usando lower() atrapa todas las apariciones de la palabra que está buscando, independientemente de su formato.

Escriba un programa que lea los archivos que encontró en el Proyecto Gutenberg y determine cuántas veces aparece la palabra 'el' en cada texto. Esta será una aproximación porque también contará palabras como 'entonces' y 'allí'.
Intente contar 'el ', con un espacio en la cadena, y vea cuánto más baja su cuenta es.

Almacenamiento de datos

Muchos de sus programas le pedirán a los usuarios que ingresen ciertos tipos de información. Puede permitir que los usuarios almacenen preferencias en un juego o proporcionen datos para una visualización. Cualquiera que sea el enfoque de su programa, almacenará la información que los usuarios proporcionen en estructuras de datos como listas y diccionarios. Cuando los usuarios cierran un programa, casi siempre querrá guardar la información que ingresaron. Una forma sencilla de hacer esto consiste en almacenar sus datos utilizando el módulo json .

El módulo json le permite volcar estructuras de datos simples de Python en un archivo y cargar los datos de ese archivo la próxima vez que se ejecute el programa. También puede usar json para compartir datos entre diferentes programas de Python. Aún mejor, el formato de datos JSON no es específico de Python, por lo que puede compartir los datos que almacena en formato JSON con personas que trabajan en muchos otros lenguajes de programación. Es un formato útil y portátil, y es fácil de aprender.

El formato JSON (Notación de objetos de JavaScript) se desarrolló originalmente para JavaScript. Sin embargo, desde entonces se ha convertido en un formato común utilizado por muchos lenguajes, incluido Python.

Usando json.dump() y json.load()

Escribamos un programa corto que almacene un conjunto de números y otro programa que vuelva a leer estos números en la memoria. El primer programa utilizará json.dump() para almacenar el conjunto de números y el segundo programa utilizará json.load().

La función json.dump() toma dos argumentos: un dato para almacenar y un objeto de archivo que puede usar para almacenar los datos. Así es como puede usar json.dump() para almacenar una lista de números:

```
número de importación json
_escritor.py
números = [2, 3, 5, 7, 11, 13]

u nombre de archivo = 'números.json'
v con abierto (nombre de archivo, 'w') como f:
w json.dump(números, f)
```

Primero importamos el módulo json y luego creamos una lista de números para trabajar. En u elegimos un nombre de archivo en el que almacenar la lista de números.

Es habitual utilizar la extensión de archivo .json para indicar que los datos del archivo se almacenan en formato JSON. Luego abrimos el archivo en modo de escritura, lo que le permite a json escribir los datos en el archivo v. En w usamos json.dump() función para almacenar los números de la lista en el archivo *numbers.json.*

Este programa no tiene salida, pero abramos el archivo *number.json* y Míralo. Los datos se almacenan en un formato que se parece a Python:

```
[2, 3, 5, 7, 11, 13]
```

Ahora escribiremos un programa que use json.load() para volver a leer la lista en la memoria:

```
número de importación json
_lector.py
u nombre de archivo = 'números.json'
v con abierto (nombre de archivo) como f:
w números = json.load(f)

imprimir (números)
```

En u nos aseguramos de leer del mismo archivo en el que escribimos. Esta vez cuando abrimos el archivo, lo abrimos en modo lectura porque Python solo necesita leer del archivo v. En w usamos la función json.load() para cargar la información almacenada en *numbers.json,* y la asignamos al numeros variables

Finalmente imprimimos la lista de números recuperada y vemos que es la misma lista creada en *number_writer.py:*

```
[2, 3, 5, 7, 11, 13]
```

Esta es una forma sencilla de compartir datos entre dos programas.

Guardar y leer datos generados por el usuario

Guardar datos con json es útil cuando trabaja con datos generados por el usuario, porque si no almacena la información de su usuario de alguna manera, la perderá cuando el programa deje de ejecutarse. Veamos un ejemplo en el que le pedimos al usuario su nombre la primera vez que ejecuta un programa y luego recordamos su nombre cuando ejecuta el programa nuevamente.

Comencemos almacenando el nombre del usuario:

recuerda importar json
_me.py

```
u nombre de usuario = entrada ("¿Cuál es tu nombre?")

nombre de archivo = 'nombre de usuario.json'
con abierto (nombre de archivo, 'w') como f:
v json.dump(nombre de usuario, f)
w print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")
```

En u solicitamos un nombre de usuario para almacenar. Luego, usamos json.dump(), pasándole un nombre de usuario y un objeto de archivo, para almacenar el nombre de usuario en un archivo v. Luego imprimimos un mensaje informando al usuario que hemos almacenado su información w:

```
¿Cómo te llamas? eric
¡Te recordaremos cuando vuelvas, Eric!
```

Ahora escribamos un nuevo programa que salude a un usuario cuyo nombre ya ha sido almacenado:

saludar_usuario.py

```
importar json

nombre de archivo = 'nombre de usuario.json'

con abierto (nombre de archivo) como f:
u nombre de usuario = json.load(f)
v print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
```

En u usamos json.load() para leer la información almacenada en *username.json* y asígnelo a la variable nombre de usuario. Ahora que hemos recuperado el nombre de usuario, podemos darle la bienvenida v:

```
¡Bienvenido de nuevo, Erick!
```

Necesitamos combinar estos dos programas en un solo archivo. Cuando alguien ejecuta *Remember_me.py,* queremos recuperar su nombre de usuario de la memoria si es posible; por lo tanto, comenzaremos con un bloque de prueba que intenta recuperar el nombre de usuario. Si el archivo *nombreusuario.json* no existe, tendremos el aviso de bloque de excepción para un nombre de usuario y lo almacenaremos en *nombreusuario.json* para la próxima vez:

```
recuerda importar json
  _me.py
        # Cargue el nombre de usuario, si se ha almacenado previamente.
        # De lo contrario, solicite el nombre de usuario y guárdelo.
        nombre de archivo = 'nombre de usuario.json'
        tratar:
      u con abierto (nombre de archivo) como f:
      en          nombre de usuario = json.load(f)
      w excepto FileNotFoundError:
      x nombre de usuario = entrada ("¿Cuál es tu nombre?")
      y con abierto (nombre de archivo, 'w') como f:
                  json.dump(nombre de usuario, f)
                  print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")
        demás:
            print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
```

No hay código nuevo aquí; los bloques de código de los dos últimos ejemplos se combinan en un solo archivo. En u intentamos abrir el archivo *username.json.* Si este archivo existe, volvemos a leer el nombre de usuario en la memoria v e imprimimos un mensaje de bienvenida al usuario en el bloque else . Si esta es la primera vez que el usuario ejecuta el programa, el *nombre* de usuario.json no existirá y aparecerá un FileNotFoundError ocurrirá w. Python pasará al bloque excepto donde le pediremos al usuario que ingrese su nombre de usuario x. Luego usamos json.dump() para almacenar el nombre de usuario e imprimir un saludo y.

Cualquiera que sea el bloque que se ejecuta, el resultado es un nombre de usuario y un apropiado saludo. Si esta es la primera vez que se ejecuta el programa, esta es la salida:

```
¿Cómo te llamas? eric
¡Te recordaremos cuando vuelvas, Eric!
```

De lo contrario:

```
¡Bienvenido de nuevo, Erick!
```

Este es el resultado que ve si el programa ya se ejecutó al menos una vez.

refactorización

A menudo, llegará a un punto en el que su código funcionará, pero reconocerá que puede mejorar el código si lo divide en una serie de funciones que tienen tareas específicas. Este proceso se llama *refactorización.* La refactorización hace que su código sea más limpio, más fácil de entender y más fácil de extender.

Podemos refactorizar *Remember_me.py* moviendo la mayor parte de su lógica a una o más funciones. El enfoque de *Remember_me.py* es saludar al usuario, así que vamos a mover todo nuestro código existente a una función llamada greeting_user():

recuerda
_me.py

```
importar json

def saludo_usuario():
u """Saluda al usuario por su nombre."""
    nombre de archivo = 'nombre de usuario.json'
    tratar:
        con abierto (nombre de archivo) como f:
            nombre de usuario = json.load(f)
    excepto FileNotFoundError:
        nombre de usuario = entrada ("¿Cuál es tu nombre?")
        con abierto (nombre de archivo, 'w') como f:
            json.dump(nombre de usuario, f)
            print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")
    demás:
        print(f"¡Bienvenido de nuevo, {nombre de usuario}!")

saludar_usuario()
```

Debido a que estamos usando una función ahora, actualizamos los comentarios con una cadena de documentación que refleja cómo funciona actualmente el programa u. Este archivo es un poco más limpio, pero la función greeting_user() hace más que solo saludar al usuario: también recupera un nombre de usuario almacenado, si existe, y solicita un nuevo nombre de usuario si no existe.

Refactoricemos greeting_user() para que no haga tantas tareas diferentes. Comenzaremos moviendo el código para recuperar un nombre de usuario almacenado a una función separada:

```
importar json

def get_stored_username():
u """Obtener nombre de usuario almacenado si está disponible."""
    nombre de archivo = 'nombre de usuario.json'
    tratar:
        con abierto (nombre de archivo) como f:
            nombre de usuario = json.load(f)
    excepto FileNotFoundError:
en      volver Ninguno
    demás:
        devolver nombre de usuario
```

```
def saludo_usuario():
    """Saluda al usuario por su nombre."""
    nombre de usuario = get_stored_username()
w si nombre de usuario:
        print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
    demás:
        nombre de usuario = entrada ("¿Cuál es tu nombre?")
        nombre de archivo = 'nombre de usuario.json'
        con abierto (nombre de archivo, 'w') como f:
            json.dump(nombre de usuario, f)
            print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")

saludar_usuario()
```

La nueva función get_stored_username() tiene un propósito claro, como se indica en la cadena de documentación en u. Esta función recupera un nombre de usuario almacenado y devuelve el nombre de usuario si encuentra uno. Si el archivo *nombreusuario.json* no existe, la función devuelve Ninguno v. Esta es una buena práctica: una función debe devolver el valor que espera o Ninguno. Esto nos permite realizar una prueba simple con el valor de retorno de la función. En w imprimimos un mensaje de bienvenida para el usuario si el intento de recuperar un nombre de usuario fue exitoso y, si no es así, solicitamos un nuevo nombre de usuario.

Deberíamos factorizar un bloque más de código fuera de greeting_user(). Si el nombre de usuario no existe, debemos mover el código que solicita un nuevo nombre de usuario a una función dedicada a ese propósito:

```
importar json

def get_stored_username():
    """Obtener nombre de usuario almacenado si está disponible."""
    --recorte--

def obtener_nuevo_nombre_de_usuario():
    """Solicitar un nuevo nombre de usuario."""
    nombre de usuario = entrada ("¿Cuál es tu nombre?")
    nombre de archivo = 'nombre de usuario.json'
    con abierto (nombre de archivo, 'w') como f:
        json.dump(nombre de usuario, f)
    devolver nombre de usuario

def saludo_usuario():
    """Saluda al usuario por su nombre."""
    nombre de usuario = get_stored_username()
    si nombre de usuario:
        print(f"¡Bienvenido de nuevo, {nombre de usuario}!")
    demás:
        nombre de usuario = get_new_username()
        print(f"¡Te recordaremos cuando regreses, {nombre de usuario}!")

saludar_usuario()
```

Cada función en esta versión final de *Remember_me.py* tiene un propósito único y claro. Llamamos a greeting_user(), y esa función imprime un mensaje apropiado: da la bienvenida a un usuario existente o saluda a un nuevo usuario. Lo hace llamando a get_stored_username(), que es responsable solo de recuperar un nombre de usuario almacenado, si existe. Finalmente, greeting_user() llama a get_new_username() si es necesario, que se encarga únicamente de obtener un nuevo nombre de usuario y almacenarlo. Esta compartimentación del trabajo es una parte esencial de la escritura de un código claro que será fácil de mantener y ampliar.

Inténtalo tú mismo

10-11. Número favorito: escriba un programa que solicite el número favorito del usuario. Use json.dump() para almacenar este número en un archivo. Escriba un programa separado que lea este valor e imprima el mensaje, "¡Sé su número favorito! Su _____."

10-12. Número favorito recordado: combine los dos programas del ejercicio 10-11 en un solo archivo. Si el número ya está almacenado, informe el número favorito al usuario. De lo contrario, solicite el número favorito del usuario y guárdelo en un archivo. Ejecute el programa dos veces para ver si funciona.

10-13. Verificar usuario: la lista final de Remember_me.py asume que el usuario ya ingresó su nombre de usuario o que el programa se está ejecutando por primera vez. Deberíamos modificarlo en caso de que el usuario actual no sea la última persona que usó el programa.

Antes de imprimir un mensaje de bienvenida en greeting_user(), pregunte al usuario si este es el nombre de usuario correcto. Si no es así, llame a get_new_username() para obtener el correcto nombre de usuario.

Resumen

En este capítulo, aprendió a trabajar con archivos. Aprendió a leer un archivo completo a la vez ya leer el contenido de un archivo una línea a la vez. Aprendió a escribir en un archivo y agregar texto al final de un archivo. Lea acerca de las excepciones y cómo manejar las excepciones que probablemente verá en sus programas. Finalmente, aprendió cómo almacenar estructuras de datos de Python para que pueda guardar la información que proporcionan sus usuarios, evitando que tengan que empezar de nuevo cada vez que ejecutan un programa.

En el Capítulo 11, aprenderá formas eficientes de probar su código. Esto ayudará confía en que el código que desarrolla es correcto y lo ayudará a identificar los errores que se introducen a medida que continúa desarrollando los programas que ha escrito.

11

Probar su código

Cuando escribe una función o una clase, también puede escribir pruebas para ese código. Las pruebas demuestran que su código funciona como se supone que debe hacerlo en respuesta a todos los tipos de entrada para los que está diseñado. Cuando escribe pruebas, puede estar seguro de que su código funcionará correctamente a medida que más personas comiencen a usar sus programas. También podrá probar código nuevo a medida que lo agrega para asegurarse de que sus cambios no rompan el comportamiento existente de su programa. Todos los programadores cometen errores, por lo que todos los programadores deben probar su código con frecuencia, detectando problemas antes de que los usuarios los encuentren.

En este capítulo, aprenderá a probar su código usando herramientas en el módulo unittest de Python. Aprenderá a crear un caso de prueba y comprobará que un conjunto de entradas da como resultado la salida que desea. Verá cómo se ve una prueba que pasa y cómo se ve una prueba que falla, y aprenderá cómo una prueba que falla puede ayudarlo a mejorar su código. Aprenderá a probar funciones y clases, y comenzará a comprender cuántas pruebas escribir para un proyecto.

Probar una función

Para aprender acerca de las pruebas, necesitamos código para probar. Aquí hay una función simple que toma un nombre y apellido, y devuelve un nombre completo con un formato limpio:

```
nombre def get_formatted_name(nombre, apellido):
_function.py """Generar un nombre completo bien formateado."""
        nombre_completo = f"{primero} {último}"
        devolver nombre_completo.título()
```

La función get_formatted_name() combina el nombre y apellido con un espacio en el medio para completar un nombre completo, y luego escribe en mayúsculas y devuelve el nombre completo. Para verificar que get_formatted_name() funciona, hagamos un programa que use esta función. El programa *names.py* permite a los usuarios ingresar un nombre y apellido, y ver un nombre completo con un formato limpio:

```
nombres.py   from name_function import get_formatted_name

print("Ingrese 'q' en cualquier momento para salir.")
mientras que es cierto:
    primero = entrada("\nPor favor, dame un nombre: ")
    si primero == 'q':
        romper
    apellido = entrada ("Por favor, dame un apellido:")
    si ultimo == 'q':
        romper

    formatted_name = get_formatted_name (primero, último)
    print(f"\tNombre bien formateado: {formatted_name}.")
```

Este programa importa get_formatted_name() desde *name_function.py.* Él el usuario puede ingresar una serie de nombres y apellidos, y ver los nombres completos con formato que se generan:

```
Introduzca 'q' en cualquier momento para salir.

Por favor, dame un nombre: janis
Por favor dame un apellido: joplin
        Nombre cuidadosamente formateado: Janis Joplin.

Por favor, dame un nombre: bob
Por favor dame un apellido: dylan
        Nombre cuidadosamente formateado: Bob Dylan.

Por favor dame un nombre: q
```

Podemos ver que los nombres generados aquí son correctos. Pero digamos que queremos modificar get_formatted_name() para que también pueda manejar segundos nombres. Mientras lo hacemos, queremos asegurarnos de no romper la forma en que la función maneja los nombres que solo tienen un nombre y un apellido. Podríamos probar nuestro código ejecutando *names.py* e ingresando un nombre como Janis Joplin cada vez que modificamos get_formatted_name(), pero eso sería tedioso. Afortunadamente,

Python proporciona una forma eficiente de automatizar la prueba de la salida de una función. Si automatizamos la prueba de get_formatted_name(), siempre podemos estar seguros de que la función funcionará cuando se le den los tipos de nombres para los que hemos escrito pruebas.

Pruebas unitarias y casos de prueba

El módulo unittest de la biblioteca estándar de Python proporciona herramientas para probar su código. Una *prueba unitaria* verifica que un aspecto específico del comportamiento de una función es correcto. Un *caso de prueba* es una colección de pruebas unitarias que juntas prueban que una función se comporta como se supone que debe hacerlo, dentro de la gama completa de situaciones que espera que maneje. Un buen caso de prueba considera todos los posibles tipos de entrada que una función podría recibir e incluye pruebas para representar cada una de estas situaciones. Un caso de prueba con *cobertura completa* incluye una gama completa de pruebas unitarias que cubren todas las formas posibles en que puede usar una función. Lograr una cobertura completa en un proyecto grande puede ser desalentador. A menudo es lo suficientemente bueno escribir pruebas para los comportamientos críticos de su código y luego apuntar a una cobertura completa solo si el proyecto comienza a tener un uso generalizado.

Una prueba de aprobación

La sintaxis para configurar un caso de prueba requiere algo de tiempo para acostumbrarse, pero una vez que haya configurado el caso de prueba, es sencillo agregar más pruebas unitarias para sus funciones. Para escribir un caso de prueba para una función, importe el módulo unittest y la función que desea probar. Luego crea una clase que herede de unittest.TestCase y escribe una serie de métodos para probar diferentes aspectos del comportamiento de tu función.

Aquí hay un caso de prueba con un método que verifica que la función get_formatted_name() funciona correctamente cuando se le da un nombre y apellido:

nombre_prueba_función.py

```
prueba unitaria de importación
from name_function import get_formatted_name

en la clase NamesTestCase (unittest.TestCase):
    """Pruebas para 'name_function.py'."""

    def test_first_last_name(self):
        """¿Funcionan nombres como 'Janis Joplin'?"""
        nombre_formateado = obtener_nombre_formateado('janis', 'joplin')
volvo   self.assertEqual(formatted_name, 'Janis Joplin')

ÿ si __nombre__ == '__principal__':
    unittest.principal()
```

Primero, importamos unittest y la función que queremos probar, get_formatted _nombre(). En u creamos una clase llamada NamesTestCase, que contendrá una serie de pruebas unitarias para get_formatted_name(). Puede nombrar la clase como desee, pero es mejor llamarla de alguna manera relacionada con la función que está a punto de probar y usar la palabra *Prueba* en el nombre de la clase. Esta clase debe heredar de la clase unittest.TestCase para que Python sepa cómo ejecutar las pruebas que escribe.

NamesTestCase contiene un solo método que prueba un aspecto de get_formatted_name(). Llamamos a este método test_first_last_name() porque estamos verificando que los nombres con solo un nombre y apellido tengan el formato correcto. Cualquier método que comience con test_ se ejecutará automáticamente cuando *ejecutemos test_name_function.py.* Dentro de este método de prueba, llamamos a la función que queremos probar. En este ejemplo llamamos a get_formatted_name() con los argumentos 'janis' y 'joplin', y asignamos el resultado a formatted_name v.

En w usamos una de las características más útiles de unittest : un método de *afirmación* . Los métodos de afirmación verifican que un resultado que recibió coincida con el resultado que esperaba recibir. En este caso, debido a que sabemos que se supone que get_formatted_name() devuelve un nombre completo en mayúsculas y correctamente espaciado, esperamos que el valor de formatted_name sea Janis Joplin. Para verificar si esto es cierto, usamos el método assertEqual() de unittest y le pasamos formatted_name y 'Janis Joplin'. La línea

```
self.assertEqual(formatted_name, 'Janis Joplin')
```

dice: “Compare el valor en formatted_name con la cadena 'Janis Joplin'. Si son iguales a lo esperado, bien. Pero si no coinciden, ¡avísame!”.

Vamos a ejecutar este archivo directamente, pero es importante tener en cuenta que muchos marcos de prueba importan sus archivos de prueba antes de ejecutarlos. Cuando se importa un archivo, el intérprete ejecuta el archivo a medida que se importa. el si El bloque en ÿ busca una variable especial, __nombre__, que se establece cuando se ejecuta el programa. Si este archivo se ejecuta como el programa principal, el valor de __name__ se establece en '__main__'. En este caso, llamamos a unittest.main(), que ejecuta el caso de prueba. Cuando un marco de prueba importa este archivo, el valor de __name__ no será '__main__' y este bloque no se ejecutará.

Cuando ejecutamos *test_name_function.py,* obtenemos el siguiente resultado:

```
.
----------------------------------------------------------------------
Ejecutó 1 prueba en 0.000s

DE ACUERDO
```

El punto en la primera línea de salida nos dice que pasó una sola prueba. La siguiente línea nos dice que Python ejecutó una prueba y tardó menos de 0,001 segundos en ejecutarse. El OK final nos dice que todas las pruebas unitarias en el caso de prueba pasaron.

Esta salida indica que la función get_formatted_name() siempre funcionará para nombres que tengan nombre y apellido a menos que modifiquemos la función. Cuando modificamos get_formatted_name(), podemos ejecutar esta prueba nuevamente. Si pasa el caso de prueba, sabemos que la función seguirá funcionando para nombres como Janis Joplin.

Una prueba fallida

¿Cómo se ve una prueba fallida? Modifiquemos get_formatted_name() para que pueda manejar segundos nombres, pero lo haremos de una manera que rompa la función para nombres con solo un nombre y apellido, como Janis Joplin.

Aquí hay una nueva versión de get_formatted_name() que requiere un argumento de segundo nombre:

```
nombre def get_formatted_name(primero, medio, último):
_function.py """Generar un nombre completo bien formateado."""
        nombre_completo = f"{primero} {medio} {último}"
        devolver nombre_completo.título()
```

Esta versión debería funcionar para personas con segundo nombre, pero cuando probamos lo, vemos que hemos roto la función para las personas con sólo un nombre y apellido. Esta vez, ejecutar el archivo *test_name_function.py* da este resultado:

```
en mi
    ================================================= ===================
v ERROR: test_first_last_name (__main__.NamesTestCase)

w Rastreo (última llamada más reciente): Archivo
      "test_name_function.py", línea 8, en test_first_last_name
        nombre_formateado = obtener_nombre_formateado('janis', 'joplin')
    TypeError: get_formatted_name() falta 1 argumento posicional requerido: 'último'

x Realizó 1 prueba en 0.000s

y FALLIDO (errores=1)
```

Hay mucha información aquí porque es posible que necesite saber mucho cuando falla una prueba. El primer elemento de la salida es un solo E u, que nos dice que una prueba unitaria en el caso de prueba resultó en un error. A continuación, vemos que test_first_last_name() en NamesTestCase causó un error v. Saber qué prueba falló es fundamental cuando su caso de prueba contiene muchas pruebas unitarias.
En w vemos un rastreo estándar, que informa que la llamada de función get_formatted_name('janis', 'joplin') ya no funciona porque falta un argumento posicional requerido.

También vemos que se ejecutó una prueba unitaria x. Finalmente, vemos un mensaje adicional que indica que el caso de prueba general falló y que ocurrió un error al ejecutar el caso de prueba y. Esta información aparece al final de la salida para que la vea de inmediato; no necesita desplazarse hacia arriba a través de una larga lista de resultados para averiguar cuántas pruebas fallaron.

Responder a una prueba fallida

¿Qué haces cuando falla una prueba? Suponiendo que está comprobando las condiciones correctas, una prueba satisfactoria significa que la función se está comportando correctamente y una prueba fallida significa que hay un error en el nuevo código que escribió. Entonces, cuando una prueba falla, no cambie la prueba. En su lugar, corrija el código que provocó que la prueba fallara. Examine los cambios que acaba de realizar en la función y descubra cómo esos cambios rompieron el comportamiento deseado.

En este caso, get_formatted_name() solía requerir solo dos parámetros: un nombre y un apellido. Ahora requiere un primer nombre, segundo nombre y

apellido. La adición de ese parámetro de segundo nombre obligatorio rompió el comportamiento deseado de get_formatted_name(). La mejor opción aquí es hacer que el segundo nombre sea opcional. Una vez que lo hagamos, nuestra prueba para nombres como Janis Joplin debería pasar nuevamente, y también deberíamos poder aceptar segundos nombres. Modifiquemos get_formatted_name() para que los segundos nombres sean opcionales y luego ejecutemos el caso de prueba nuevamente. Si pasa, nos aseguraremos de que la función maneje los segundos nombres correctamente.

Para que los segundos nombres sean opcionales, movemos el parámetro medio al final de la lista de parámetros en la definición de la función y asígnele un valor predeterminado vacío. También agregamos una prueba if que genera el nombre completo correctamente, dependiendo de si se proporciona o no un segundo nombre:

nombre _function.py

```
def get_formatted_name(nombre, apellido, medio=''):
    """Generar un nombre completo bien formateado."""
    si medio:
        nombre_completo = f"{primero} {medio} {último}"
    demás:
        nombre_completo = f"{primero} {último}"
    devolver nombre_completo.título()
```

En esta nueva versión de get_formatted_name(), el segundo nombre es opcional. Si se pasa un segundo nombre a la función, el nombre completo contendrá un nombre, un segundo nombre y un apellido. De lo contrario, el nombre completo constará solo de un nombre y apellido. Ahora la función debería funcionar para ambos tipos de nombres. Para averiguar si la función aún funciona para nombres como Janis Joplin, *ejecutemos test_name_function.py* nuevamente :

```
.
----------------------------------------------------------------------
Ejecutó 1 prueba en 0.000s

DE ACUERDO
```

El caso de prueba pasa ahora. Esto es ideal; significa que la función vuelve a funcionar para nombres como Janis Joplin sin que tengamos que probar la función manualmente. Arreglar nuestra función fue fácil porque la prueba fallida nos ayudó a identificar el nuevo código que rompía el comportamiento existente.

Adición de nuevas pruebas

Ahora que sabemos que get_formatted_name() funciona nuevamente para nombres simples, escribamos una segunda prueba para las personas que incluyen un segundo nombre. Hacemos esto agregando otro método a la clase NamesTestCase:

nombre_prueba_función.py

```
--recorte--

clase NombresTestCase (unittest.TestCase):
    """Pruebas para 'name_function.py'."""

    def test_first_last_name(self):
        --recorte--
```

```
    def test_first_last_middle_name(self):
        """¿Funcionan nombres como 'Wolfgang Amadeus Mozart'?"""
en      formatted_name = get_formatted_name(
            'wolfgang', 'mozart', 'amadeus')
        self.assertEqual(formatted_name, 'Wolfgang Amadeus Mozart')

si __nombre__ == '__principal__':
    unittest.principal()
```

Llamamos a este nuevo método test_first_last_middle_name(). El nombre del método debe comenzar con *test_* para que el método se ejecute automáticamente cuando ejecutamos *test_name_function.py.* Nombramos el método para dejar claro qué comportamiento de get_formatted_name() estamos probando. Como resultado, si la prueba falla, sabemos de inmediato qué tipo de nombres se ven afectados. Está bien tener nombres de métodos largos en sus clases de TestCase . Deben ser descriptivos para que pueda entender el resultado cuando fallan las pruebas y, dado que Python los llama automáticamente, nunca tendrá que escribir código que llame a estos métodos.

Para probar la función, llamamos a get_formatted_name() con un nombre, segundo y segundo nombre u, y luego usamos assertEqual() para verificar que el nombre completo devuelto coincida con el nombre completo (primero, segundo y último) que esperamos .
Cuando ejecutamos *test_name_function.py* nuevamente, ambas pruebas pasan:

```
..
----------------------------------------------------------------------
Realizó 2 pruebas en 0.000s

DE ACUERDO
```

¡Estupendo! Ahora sabemos que la función aún funciona para nombres como Janis Joplin, y podemos estar seguros de que también funcionará para nombres como Wolfgang Amadeus Mozart .

Inténtalo tú mismo

11-1. Ciudad, País: Escriba una función que acepte dos parámetros: un nombre de ciudad y un nombre de país. La función debe devolver una única cadena con el formato *Ciudad, País,* como Santiago, Chile. Almacene la función en un módulo llamado city_functions.py.

Cree un archivo llamado test_cities.py que pruebe la función que acaba de escribir (recuerde que necesita importar unittest y la función que desea probar). Escriba un método llamado test_city_country() para verificar que llamar a su función con valores como 'santiago' y 'chile' da como resultado la cadena correcta. Ejecute test_cities.py y asegúrese de que test_city_country() pase.

(continuado)

11-2. Población: modifique su función para que requiera un tercer parámetro, la población. Ahora debería devolver una sola cadena de la forma *Ciudad, País – población xxx,* como Santiago, Chile – población 5000000. Ejecutar prueba _cities.py de nuevo. Asegúrate de que test_city_country() falle esta vez.

Modifique la función para que el parámetro de población sea opcional. Ejecutar prueba _cities.py nuevamente, y asegúrese de que test_city_country() vuelva a pasar.

Escriba una segunda prueba llamada test_city_country_population() que verifique que puede llamar a su función con los valores 'santiago', 'chile' y 'population=5000000'. Vuelva a ejecutar test_cities.py y asegúrese de que pase esta nueva prueba.

Probar una clase

En la primera parte de este capítulo, escribió pruebas para una sola función. Ahora escribirá pruebas para una clase. Utilizará clases en muchos de sus propios programas, por lo que es útil poder probar que sus clases funcionan correctamente. Si tiene exámenes aprobados para una clase en la que está trabajando, puede estar seguro de que las mejoras que realice en la clase no romperán accidentalmente su comportamiento actual.

Una variedad de métodos de afirmación

Python proporciona una serie de métodos de afirmación en la clase unittest.TestCase . Como se mencionó anteriormente, los métodos de afirmación prueban si una condición que cree que es verdadera en un punto específico de su código es realmente verdadera. Si la condición es verdadera como se esperaba, se confirma su suposición sobre cómo se comporta esa parte de su programa; puede estar seguro de que no existen errores. Si la condición que supone que es verdadera en realidad no lo es, Python genera una excepción.

La tabla 11-1 describe seis métodos de afirmación comúnmente utilizados. Con estos métodos, puede verificar que los valores devueltos sean iguales o no iguales a los valores esperados, que los valores sean verdaderos o falsos, y que los valores estén o no en una lista determinada. Puede usar estos métodos solo en una clase que hereda de unittest .TestCase, así que veamos cómo podemos usar uno de estos métodos en el contexto de probar una clase real.

Tabla 11-1: Métodos de afirmación disponibles desde el módulo unittest

Método	Usar
afirmarIgual(a, b)	Verifique que a == b
afirmarNoEqual(a, b)	Verifica que a != b
afirmarVerdadero(x)	Verifique que x sea Verdadero
afirmarFalso(x)	Verifica que x es Falso
afirmarEn(elemento, *lista)*	Verifique que el *artículo* esté en la *lista*
afirmarNoEn(elemento, *lista)*	Verifique que el *artículo* no esté en la *lista*

Una clase para probar

Probar una clase es similar a probar una función: gran parte de su trabajo consiste en probar el comportamiento de los métodos de la clase. Pero hay algunas diferencias, así que escribamos una clase para probar. Considere una clase que ayude a administrar encuestas anónimas:

encuesta.py

```
clase Encuesta anónima:
    """Recopilar respuestas anónimas a una pregunta de encuesta."""

u def __init__(uno mismo, pregunta):
        """Guarde una pregunta y prepárese para almacenar respuestas."""
        self.pregunta = pregunta
        auto.respuestas = []

v def mostrar_pregunta(auto):
        """Mostrar la pregunta de la encuesta."""
        imprimir(auto.pregunta)

w def store_response(self, new_response):
        """Almacenar una única respuesta a la encuesta."""
        self.responses.append(nueva_respuesta)

x def mostrar_resultados(uno mismo):
        """Mostrar todas las respuestas que se han dado."""
        imprimir("Resultados de la encuesta:")
        para respuesta en self.responses:
            imprimir(f"- {respuesta}")
```

Esta clase comienza con una pregunta de encuesta que usted proporciona e incluye una lista vacía para almacenar las respuestas. La clase tiene métodos para imprimir la pregunta de la encuesta v, agregar una nueva respuesta a la lista de respuestas w e imprimir todas las respuestas almacenadas en la lista x. Para crear una instancia de esta clase, todo lo que tiene que proporcionar es una pregunta. Una vez que tenga una instancia que represente una encuesta en particular, muestre la pregunta de la encuesta con show_question(), almacene una respuesta usando store_response() y muestre los resultados con show_results().

Para mostrar que la clase EncuestaAnónima funciona, escribamos un programa que use la clase:

idioma _encuesta.py

```
desde la importación de encuestas AnonymousSurvey

# Definir una pregunta y hacer una encuesta.
pregunta = "¿Qué idioma aprendiste a hablar primero?"
my_survey = AnonymousSurvey(pregunta)

# Muestra la pregunta y almacena las respuestas a la pregunta.
mi_encuesta.mostrar_pregunta()
print("Ingrese 'q' en cualquier momento para salir.\n")
mientras que es cierto:
    respuesta = entrada ("Idioma:")
    si respuesta == 'q':
        romper
    my_survey.store_response(respuesta)
```

```
# Mostrar los resultados de la encuesta.
print("\n¡Gracias a todos los que participaron en la encuesta!")
mi_encuesta.mostrar_resultados()
```

Este programa define una pregunta ("¿Qué idioma aprendiste a hablar primero?") y crea un objeto AnonymousSurvey con esa pregunta. El programa llama a show_question() para mostrar la pregunta y luego solicita respuestas. Cada respuesta se almacena a medida que se recibe. Cuando se han ingresado todas las respuestas (el usuario ingresa q para salir), show_results() imprime los resultados de la encuesta:

```
¿Qué idioma aprendiste a hablar primero?
Introduzca 'q' en cualquier momento para salir.

Idioma: Inglés
Idioma: Español
Idioma: Inglés
Idioma: mandarín
Idioma: q

¡Gracias a todos los que participaron en la encuesta!
Resultados de la
encuesta: - Inglés -
Español - Inglés

- mandarín
```

Esta clase funciona para una simple encuesta anónima. Pero digamos que queremos mejorar AnonymousSurvey y el módulo en el que se encuentra, encuesta. Podríamos permitir que cada usuario ingrese más de una respuesta. Podríamos escribir un método para enumerar solo respuestas únicas y para informar cuántas veces se dio cada respuesta. Podríamos escribir otra clase para gestionar encuestas no anónimas.

Implementar tales cambios correría el riesgo de afectar el comportamiento actual. de la clase EncuestaAnónima. Por ejemplo, es posible que al intentar permitir que cada usuario ingrese varias respuestas, accidentalmente podamos cambiar la forma en que se manejan las respuestas individuales. Para asegurarnos de no interrumpir el comportamiento existente a medida que desarrollamos este módulo, podemos escribir pruebas para la clase.

Prueba de la clase AnonymousSurvey

Escribamos una prueba que verifique un aspecto del comportamiento de AnonymousSurvey . Escribiremos una prueba para verificar que una sola respuesta a la pregunta de la encuesta se almacene correctamente. Usaremos el método assertIn() para verificar que la respuesta esté en la lista de respuestas después de que se haya almacenado:

prueba
_encuesta.py

```
prueba unitaria de importación
desde la importación de encuestas AnonymousSurvey

u class TestAnonymousSurvey(unittest.TestCase):
    """Pruebas para la clase AnonymousSurvey"""
```

```
v def test_store_single_response(auto):
            """Prueba que una sola respuesta se almacena correctamente."""
            pregunta = "¿Qué idioma aprendiste a hablar primero?"
en          my_survey = AnonymousSurvey(pregunta)
            my_survey.store_response('Inglés')
X           self.assertIn('Inglés', mi_encuesta.respuestas)

  si __nombre__ == '__principal__':
      unittest.principal()
```

Comenzamos importando el módulo unittest y la clase que queremos probar, AnonymousSurvey. Llamamos a nuestro caso de prueba TestAnonymousSurvey, que nuevamente hereda de unittest.TestCase u. El primer método de prueba verificará que cuando almacenamos una respuesta a la pregunta de la encuesta, la respuesta termina en la lista de respuestas de la encuesta. Un buen nombre descriptivo para este método es test_store_single_response() v. Si esta prueba falla, sabremos por el nombre del método que se muestra en el resultado de la prueba fallida que hubo un problema al almacenar una sola respuesta a la encuesta.

Para probar el comportamiento de una clase, necesitamos crear una instancia de la clase. En w creamos una instancia llamada my_survey con la pregunta "¿Qué idioma aprendiste a hablar por primera vez?" Almacenamos una sola respuesta, en inglés, usando el método store_response() . Luego verificamos que la respuesta se almacenó correctamente afirmando que el inglés está en la lista my_survey.responses x.

Cuando ejecutamos *test_survey.py,* la prueba pasa:

```
.
----------------------------------------------------------------------
Ejecutó 1 prueba en 0.001 s

DE ACUERDO
```

Esto es bueno, pero una encuesta es útil solo si genera más de una respuesta. Verifiquemos que tres respuestas se pueden almacenar correctamente. Para hacer esto, agregamos otro método a TestAnonymousSurvey:

```
prueba unitaria de importación
desde la importación de encuestas AnonymousSurvey

clase TestAnonymousSurvey(unittest.TestCase):
    """Pruebas para la clase AnonymousSurvey"""

    def test_store_single_response(auto):
        --recorte--

    def test_store_tres_respuestas(uno mismo):
        """Pruebe que tres respuestas individuales se almacenan correctamente."""
        pregunta = "¿Qué idioma aprendiste a hablar primero?"
        my_survey = AnonymousSurvey(pregunta)
en      respuestas = ['Inglés', 'Español', 'Mandarín']
        para respuesta en respuestas:
            my_survey.store_response(respuesta)

en      para respuesta en respuestas:
```

```
            self.assertIn(respuesta, mi_encuesta.respuestas)

si __nombre__ == '__principal__':
    unittest.principal()
```

Llamamos al nuevo método test_store_three_responses(). Creamos un objeto de encuesta tal como lo hicimos en test_store_single_response(). Definimos una lista que contiene tres respuestas diferentes u, y luego llamamos a store_response() para cada una de estas respuestas. Una vez que se han almacenado las respuestas, escribimos otro ciclo y afirmamos que cada respuesta ahora está en my_survey.responses v.

Cuando volvemos a ejecutar *test_survey.py* , ambas pruebas (para una sola respuesta y para tres respuestas) pasar:

```
..
----------------------------------------------------------------------
Realizó 2 pruebas en 0.000s

DE ACUERDO
```

Esto funciona perfectamente. Sin embargo, estas pruebas son un poco repetitivas, por lo que use otra característica de unittest para hacerlos más eficientes.

El método de configuración ()

En *test_survey.py* creamos una nueva instancia de AnonymousSurvey en cada método de prueba y creamos nuevas respuestas en cada método. El unittest.TestCase class tiene un método setUp() que le permite crear estos objetos una vez y luego usarlos en cada uno de sus métodos de prueba. Cuando incluyes un setUp() en una clase TestCase , Python ejecuta el método setUp() antes de ejecutar cada método que comienza con *test_*. Todos los objetos creados en el método setUp() estarán disponibles en cada método de prueba que escriba.

Usemos setUp() para crear una instancia de encuesta y un conjunto de respuestas que se pueden usar en test_store_single_response() y test_store_tres_responses():

```
prueba unitaria de importación
desde la importación de encuestas AnonymousSurvey

clase TestAnonymousSurvey(unittest.TestCase):
    """Pruebas para la clase EncuestaAnónima."""

    def configurar(auto):
        """
        Cree una encuesta y un conjunto de respuestas para usar en todos los métodos de prueba.
        """
        pregunta = "¿Qué idioma aprendiste a hablar primero?"
        self.my_survey = AnonymousSurvey(pregunta)
ultravioleta        self.responses = ['Inglés', 'Español', 'Mandarín']

    def test_store_single_response(auto):
        """Prueba que una sola respuesta se almacena correctamente."""
        self.my_survey.store_response(self.responses[0])
        self.assertIn(self.responses[0], self.my_survey.responses)
```

```
    def test_store_tres_respuestas(uno mismo):
        """Pruebe que tres respuestas individuales se almacenan correctamente."""
        para respuesta en self.responses:
            self.my_survey.store_response(respuesta)
        para respuesta en self.responses:
            self.assertIn(respuesta, self.my_survey.responses)

si __nombre__ == '__principal__':
    unittest.principal()
```

El método setUp() hace dos cosas: crea una instancia de encuesta u y crea una lista de respuestas v. Cada una de estas tiene el prefijo self, por lo que pueden usarse en cualquier parte de la clase. Esto simplifica los dos métodos de prueba, porque ninguno tiene que hacer una instancia de encuesta o una respuesta.

El método test_store_single_response() verifica que la primera respuesta en self.responses— self.responses[0]—se puede almacenar correctamente y test_store _tres_respuestas() verifica que las tres respuestas en self.responses se puedan almacenar correctamente.

Cuando ejecutamos *test_survey.py* nuevamente, ambas pruebas aún pasan. Estas pruebas serían particularmente útiles cuando se intente expandir AnonymousSurvey para manejar múltiples respuestas para cada persona. Después de modificar el código para aceptar varias respuestas, puede ejecutar estas pruebas y asegurarse de que no haya afectado la capacidad de almacenar una sola respuesta o una serie de respuestas individuales.

Cuando estás probando tus propias clases, el método setUp() puede hacer que tus métodos de prueba sean más fáciles de escribir. Crea un conjunto de instancias y atributos en setUp() y luego usa estas instancias en todos sus métodos de prueba. Esto es mucho más fácil que crear un nuevo conjunto de instancias y atributos en cada método de prueba.

N ota *ando se ejecuta un caso de prueba, Python imprime un carácter para cada prueba unitaria tal como está. terminado. Una prueba que pasa imprime un punto, una prueba que da como resultado un error imprime una E y una prueba que da como resultado una aserción fallida imprime una F. Es por eso que verá una cantidad diferente de puntos y caracteres en la primera línea de salida cuando ejecuta sus casos de prueba. Si un caso de prueba tarda mucho en ejecutarse porque contiene muchas pruebas unitarias, puede ver estos resultados para tener una idea de cuántas pruebas están pasando.*

Inténtalo tú mismo

11-3. Empleado: Escribe una clase llamada Empleado. El método __init__() debe tomar un nombre, un apellido y un salario anual, y almacenar cada uno de estos como atributos. Escriba un método llamado give_raise() que agregue $5,000 al salario anual de manera predeterminada, pero que también acepte una cantidad de aumento diferente.

Escriba un caso de prueba para Empleado. Escriba dos métodos de prueba, test_give_default _raise() y test_give_custom_raise(). Utilice el método setUp() para no tener que crear una nueva instancia de empleado en cada método de prueba. Ejecute su caso de prueba y asegúrese de que ambas pruebas pasen.

Resumen

En este capítulo aprendió a escribir pruebas para funciones y clases usando herramientas en el módulo unittest . Aprendió a escribir una clase que hereda de unittest.TestCase, y aprendió a escribir métodos de prueba que verifican comportamientos específicos que deben exhibir sus funciones y clases. Aprendió a usar el método setUp() para crear de manera eficiente instancias y atributos de sus clases que se pueden usar en todos los métodos de prueba para una clase.

Las pruebas son un tema importante que muchos principiantes no aprenden. No tienes que escribir pruebas para todos los proyectos simples que intentas como principiante. Pero tan pronto como comience a trabajar en proyectos que impliquen un esfuerzo de desarrollo significativo, debe probar los comportamientos críticos de sus funciones y clases. Estará más seguro de que el nuevo trabajo en su proyecto no romperá las partes que funcionan, y esto le dará la libertad de realizar mejoras en su código. Si accidentalmente interrumpe la funcionalidad existente, lo sabrá de inmediato, por lo que aún puede solucionar el problema fácilmente. Responder a una prueba fallida que ejecutó es mucho más fácil que responder a un informe de error de un usuario descontento.

Otros programadores respetan más tus proyectos si incluyes algunas pruebas iniciales. Se sentirán más cómodos experimentando con su código y estarán más dispuestos a trabajar con usted en proyectos. Si desea contribuir a un proyecto en el que están trabajando otros programadores, se espera que demuestre que su código pasa las pruebas existentes y, por lo general, se espera que escriba pruebas para el nuevo comportamiento que introduzca en el proyecto.

Experimente con las pruebas para familiarizarse con el proceso de probar su código. Escriba pruebas para los comportamientos más críticos de sus funciones y clases, pero no apunte a una cobertura completa en los primeros proyectos a menos que tenga una razón específica para hacerlo.

Parte II

Proyectos

¡Felicidades! Ahora sabe lo suficiente sobre Python para comenzar a crear proyectos interactivos y significativos. La creación de sus propios proyectos le enseñará nuevas habilidades y consolidará su comprensión de los conceptos presentados en la Parte I.

La Parte II contiene tres tipos de proyectos, y puede elegir hacer cualquiera o todos estos proyectos en el orden que desee. Aquí hay una breve descripción de cada proyecto para ayudarlo a decidir cuál profundizar primero.

Alien Invasion: Hacer un juego con Python

En el proyecto *Alien Invasion* (Capítulos 12, 13 y 14), usarás el paquete Pygame para desarrollar un juego en 2D en el que el objetivo es derribar una flota de alienígenas mientras descienden por la pantalla en niveles que aumentan en velocidad y dificultad. Al final del proyecto, habrás aprendido habilidades que te permitirán desarrollar tus propios juegos 2D en Pygame.

Visualización de datos

El proyecto de Visualización de datos comienza en el Capítulo 15, en el que aprenderá a generar datos y crear una serie de visualizaciones funcionales y hermosas de esos datos usando Matplotlib y Plotly. El Capítulo 16 le enseña a acceder a datos de fuentes en línea y a introducirlos en un paquete de visualización para crear gráficos de datos meteorológicos y un mapa de la actividad sísmica global. Finalmente, el Capítulo 17 le muestra cómo escribir un programa para descargar automáticamente

y visualizar datos. Aprender a hacer visualizaciones te permite explorar el campo de la minería de datos, que es una habilidad muy buscada en el mundo actual.

Aplicaciones

web En el proyecto Aplicaciones web (capítulos 18, 19 y 20), utilizará el paquete Django para crear una aplicación web sencilla que permita a los usuarios llevar un diario sobre cualquier número de temas sobre los que hayan estado aprendiendo. Los usuarios crearán una cuenta con un nombre de usuario y una contraseña, ingresarán un tema y luego harán entradas sobre lo que están aprendiendo. También aprenderá cómo implementar su aplicación para que cualquier persona en el mundo pueda acceder a ella.

Después de completar este proyecto, podrá comenzar a crear sus propias aplicaciones web simples y estará listo para profundizar en recursos más completos sobre la creación de aplicaciones con Django.

Proyecto 1

Invasión alienígena



12

Un barco que dispara balas

¡Construyamos un juego llamado *Alien Invasion!* Usaremos Pygame, una colección de divertidos y potentes módulos de Python que administran gráficos, animaciones e incluso sonido, lo que facilita la creación de juegos sofisticados. Con las tareas de manejo de Pygame, como dibujar imágenes en la pantalla, puede concentrarse en la lógica de nivel superior de la dinámica del juego.

En este capítulo, configurará Pygame y luego creará un cohete espacial que se mueve de derecha a izquierda y dispara balas en respuesta a la entrada del jugador. En los próximos dos capítulos, crearás una flota de alienígenas para destruir y luego continuarás refinando el juego estableciendo límites en la cantidad de barcos que puedes usar y agregando un marcador.

Mientras crea este juego, también aprenderá a administrar proyectos grandes que abarcan varios archivos. Refactorizaremos una gran cantidad de código y administraremos el contenido de los archivos para organizar el proyecto y hacer que el código sea eficiente.

Hacer juegos es una forma ideal de divertirse mientras se aprende un idioma. Es profundamente satisfactorio jugar un juego que escribiste, y escribir un juego simple te ayudará a comprender cómo los profesionales desarrollan juegos. A medida que avanza en este capítulo, ingrese y ejecute el código para identificar cómo cada bloque de código contribuye al juego en general. Experimente con diferentes valores y configuraciones para comprender mejor cómo refinar las interacciones en sus juegos.

en cuenta que Alien Invasion *abarca varios archivos diferentes, así que cree un nuevo* alien_invasion *carpeta en su sistema. Asegúrese de guardar todos los archivos del proyecto en esta carpeta para que sus* declaraciones de *importación funcionen correctamente.*

Además, si se siente cómodo usando el control de versiones, es posible que desee usarlo para esto. proyecto. Si no ha utilizado el control de versiones antes, consulte el Apéndice D para obtener una descripción general.

Planificación de su proyecto

Cuando está construyendo un proyecto grande, es importante preparar un plan antes de comenzar a escribir código. Su plan lo mantendrá enfocado y hará que sea más probable que complete el proyecto.

Escribamos una descripción del juego general. Aunque el seguimiento Aunque esta descripción no cubre todos los detalles de *Alien Invasion,* proporciona una idea clara de cómo empezar a construir el juego:

> En *Alien Invasion,* el jugador controla un cohete espacial que aparece en la parte inferior central de la pantalla. El jugador puede mover la nave hacia la derecha y hacia la izquierda con las teclas de flecha y disparar balas con la barra espaciadora. Cuando comienza el juego, una flota de extraterrestres llena el cielo y se mueve por la pantalla. El jugador dispara y destruye a los alienígenas. Si el jugador dispara a todos los alienígenas, aparece una nueva flota que se mueve más rápido que la flota anterior. Si algún alienígena golpea la nave del jugador o llega al fondo de la pantalla, el jugador pierde una nave. Si el jugador pierde tres barcos, el juego termina.

Para la primera fase de desarrollo, crearemos una nave que pueda moverse hacia la derecha y hacia la izquierda y que dispare balas cuando el jugador presione la barra espaciadora. Después de configurar este comportamiento, podemos crear los alienígenas y refinar el juego.

Instalando Pygame

Antes de comenzar a programar, instale Pygame. El módulo pip lo ayuda a descargar e instalar paquetes de Python. Para instalar Pygame, ingrese el siguiente comando en un indicador de terminal:

```
$ python -m instalación pip --user pygame
```

Este comando le dice a Python que ejecute el módulo pip e instale el pygame paquete a la instalación de Python del usuario actual. Si utiliza un comando

que no sea python para ejecutar programas o iniciar una sesión de terminal, como python3, su comando se verá así:

```
$ python3 -m instalación pip --user pygame
```

Si este comando no funciona en macOS, intente ejecutar el comando nuevamente sin el indicador *--user .*

Iniciar el proyecto del juego

Comenzaremos a construir el juego creando una ventana de Pygame vacía. Más tarde, dibujaremos los elementos del juego, como la nave y los alienígenas, en esta ventana. También haremos que nuestro juego responda a la entrada del usuario, estableceremos el color de fondo y cargaremos una imagen de barco.

Creación de una ventana de Pygame y respuesta a la entrada del usuario

Crearemos una ventana de Pygame vacía creando una clase para representar el juego. En su editor de texto, cree un nuevo archivo y guárdelo como *alien_invasion.py;* luego ingresa lo siguiente:

alien_invasion.py

```
sistema de importación

importar pygame

clase invasión alienígena:
    """Clase general para administrar los activos y el comportamiento del juego"."""

    def __init__(uno mismo):
        """Inicializa el juego y crea recursos del juego."""
en      pygame.init ()

en      self.pantalla = pygame.display.set_mode((1200, 800))
        pygame.display.set_caption("Invasión alienígena")

    def run_game(self):
        """Iniciar el bucle principal del juego."""
en      mientras que es cierto:
            # Esté atento a los eventos del teclado y el mouse.
xy          para evento en pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()

            # Hacer visible la pantalla dibujada más recientemente.
con         pygame.display.flip()

si __nombre__ == '__principal__':
    # Cree una instancia de juego y ejecute el juego.
    ai = invasión alienígena ()
    ai.run_game()
```

Primero, importamos los módulos sys y pygame . El módulo pygame contiene la funcionalidad que necesitamos para hacer un juego. Usaremos herramientas en el sistema módulo para salir del juego cuando el jugador sale.

Alien Invasion comienza como una clase llamada AlienInvasion. En el __init__() método, la función pygame.init() inicializa la configuración de fondo que Pygame necesita para funcionar correctamente u. En v, llamamos a pygame.display.set_mode() para crear una ventana de visualización en la que dibujaremos todos los elementos gráficos del juego. El argumento (1200, 800) es una tupla que define las dimensiones de la ventana del juego, que será de 1200 píxeles de ancho por 800 píxeles de alto. (Puede ajustar estos valores según el tamaño de su pantalla). Asignamos esta ventana de visualización al atributo self.screen, por lo que estará disponible en todos los métodos de la clase.

El objeto que asignamos a self.screen se llama *superficie.* Una superficie en Pygame es una parte de la pantalla donde se puede mostrar un elemento del juego. Cada elemento del juego, como un extraterrestre o una nave, es su propia superficie. La superficie devuelta por display.set_mode() representa toda la ventana del juego. Cuando activamos el bucle de animación del juego, esta superficie se volverá a dibujar en cada pasada por el bucle, por lo que puede actualizarse con cualquier cambio desencadenado por la entrada del usuario.

El juego está controlado por el método run_game() . Este método contiene un ciclo while w que se ejecuta continuamente. El bucle while contiene un bucle de eventos y un código que gestiona las actualizaciones de pantalla. Un *evento* es una acción que el usuario realiza mientras juega, como presionar una tecla o mover el mouse. Para hacer que nuestro programa responda a los eventos, escribimos este *ciclo* de eventos para *escuchar* eventos y realizar las tareas apropiadas dependiendo de los tipos de eventos que ocurran. El bucle for en x es un bucle de eventos.

Para acceder a los eventos que detecta Pygame, usaremos el pygame.event Función .get() . Esta función devuelve una lista de eventos que han tenido lugar desde la última vez que se llamó a esta función. Cualquier evento de teclado o mouse hará que se ejecute este bucle for . Dentro del ciclo, escribiremos una serie de if declaraciones para detectar y responder a eventos específicos. Por ejemplo, cuando el jugador hace clic en el botón de cierre de la ventana del juego, se detecta un evento pygame.QUIT y llamamos a sys.exit() para salir del juego y.

La llamada a pygame.display.flip() en z le dice a Pygame que haga visible la pantalla dibujada más recientemente. En este caso, simplemente dibuja una pantalla vacía en cada paso a través del bucle while , borrando la pantalla anterior para que solo se vea la nueva pantalla. Cuando movemos los elementos del juego, pygame.display .flip() actualiza continuamente la pantalla para mostrar las nuevas posiciones de los elementos del juego y oculta las antiguas, creando la ilusión de un movimiento fluido.

Al final del archivo, creamos una instancia del juego y luego llamamos ejecutar_juego(). Colocamos run_game() en un bloque if que solo se ejecuta si se llama directamente al archivo. Cuando ejecute este archivo *alien_invasion.py* , debería ver una ventana de Pygame vacía.

Configuración del color de fondo

Pygame crea una pantalla negra por defecto, pero eso es aburrido. Establezcamos un color de fondo diferente. Haremos esto al final del método __init__() .

alien_invasion.py

```
    def __init__(uno mismo):
        --recorte--
        pygame.display.set_caption("Invasión alienígena")

        # Establecer el color de fondo.
en      self.bg_color = (230, 230, 230)

    def run_game(self):
        --recorte--
            para evento en pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()

            # Vuelva a dibujar la pantalla durante cada pasada por el bucle.
en          self.screen.fill(self.bg_color)

            # Hacer visible la pantalla dibujada más recientemente.
            pygame.display.flip()
```

Los colores en Pygame se especifican como colores RGB: una mezcla de rojo, verde y azul. Cada valor de color puede variar de 0 a 255. El valor de color (255, 0, 0) es rojo, (0, 255, 0) es verde y (0, 0, 255) es azul. Puede mezclar diferentes valores RGB para crear hasta 16 millones de colores. El valor de color (230, 230, 230) mezcla cantidades iguales de rojo, azul y verde, lo que produce un color de fondo gris claro. Asignamos este color a self.bg_color u.

En v, llenamos la pantalla con el color de fondo usando el método fill() método, que actúa sobre una superficie y toma sólo un argumento: un color.

Creación de una clase de configuración

Cada vez que introducimos una nueva funcionalidad en el juego, normalmente también creamos algunas configuraciones nuevas. En lugar de agregar configuraciones a lo largo del código, escribamos un módulo llamado configuración que contenga una clase llamada Configuración para almacenar todos estos valores en un solo lugar. Este enfoque nos permite trabajar con un solo objeto de configuración cada vez que necesitamos acceder a una configuración individual. Esto también facilita la modificación de la apariencia y el comportamiento del juego a medida que crece nuestro proyecto: para modificar el juego, simplemente cambiaremos algunos valores en *settings.py,* que crearemos a continuación, en lugar de buscar diferentes configuraciones a lo largo del proyecto. .

Cree un nuevo archivo llamado *settings.py* dentro de su carpeta *alien_invasion* y agregue esta clase de configuración inicial :

configuración.py

```
Configuración de clase:
    """Una clase para almacenar todos los ajustes de Alien Invasion."""

    def __init__(uno mismo):
        """Inicializa la configuración del juego."""
        # Ajustes de pantalla
        self.screen_width = 1200
        self.screen_height = 800
        self.bg_color = (230, 230, 230)
```

Para crear una instancia de Configuración en el proyecto y usarla para acceder a nuestro configuración, necesitamos modificar *alien_invasion.py* de la siguiente manera:

alien_invasion.py

```
--recorte--
importar pygame

desde ajustes importar Ajustes

clase invasión alienígena:
    """Clase general para administrar los activos y el comportamiento del juego".""

    def __init__(uno mismo):
        """Inicializa el juego y crea recursos del juego."""
        pygame.init ()
en      self.settings = Ajustes()

en      self.pantalla = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Invasión alienígena")

    def run_game(self):
            --recorte--
            # Vuelva a dibujar la pantalla durante cada pasada por el bucle.
en          self.screen.fill(self.settings.bg_color)

            # Hacer visible la pantalla dibujada más recientemente.
            pygame.display.flip()
--recorte--
```

Importamos la configuración al archivo principal del programa. Luego creamos una instancia de Settings y la asignamos a self.settings u, luego de hacer la llamada a pygame.init(). Cuando creamos una pantalla v, usamos los atributos screen_width y screen_height de self.settings, y luego usamos self.settings para acceder al color de fondo cuando llenamos la pantalla en w también.

Cuando *ejecutes alien_invasion.py* ahora, aún no verás ningún cambio, porque todo lo que hemos hecho es mover la configuración que ya estábamos usando en otro lugar. Ahora estamos listos para comenzar a agregar nuevos elementos a la pantalla.

Agregar la imagen del barco

Agreguemos el barco a nuestro juego. Para dibujar la nave del jugador en la pantalla, cargaremos una imagen y luego usaremos el método Pygame blit() para dibujar la imagen.

Cuando elija ilustraciones para sus juegos, asegúrese de prestar atención a Licencia. La forma más segura y económica de comenzar es usar ics gráficos con licencia libre que puede usar y modificar, desde un sitio web como *https:// pixabay.com/.*

Puede usar casi cualquier tipo de archivo de imagen en su juego, pero es más fácil cuando usa un archivo de mapa de bits *(.bmp)* porque Pygame carga mapas de bits de manera predeterminada. Aunque puede configurar Pygame para usar otros tipos de archivos, algunos tipos de archivos

dependen de ciertas bibliotecas de imágenes que deben estar instaladas en su computadora. La mayoría de las imágenes que encontrará están en formato .jpg o .png , pero puede convertirlas en mapas de bits usando herramientas como Photoshop, GIMP y Paint.

Preste especial atención al color de fondo de la imagen elegida. Intente encontrar un archivo con un fondo transparente o sólido que pueda reemplazar con cualquier color de fondo usando un editor de imágenes. Sus juegos se verán mejor si el color de fondo de la imagen coincide con el color de fondo de su juego. Alternativamente, puede hacer coincidir el fondo de su juego con el fondo de la imagen.

Para *Alien Invasion,* puede usar el archivo *ship.bmp* (Figura 12-1), que está disponible en los recursos del libro en *https:// nostarch.com/ pythoncrashcourse2e/.* El color de fondo del archivo coincide con la configuración que estamos usando en este proyecto. Cree una carpeta llamada *imágenes* dentro de su carpeta principal del proyecto *alien_invasion .* Guarde el archivo *ship.bmp* en la carpeta de *imágenes .*

Figura 12-1: La nave de Alien Invasion

Crear la clase de barco

Después de elegir una imagen para el barco, debemos mostrarla en la pantalla. Para usar nuestra nave, crearemos un nuevo módulo de nave que contendrá la clase Nave. Esta clase gestionará la mayor parte del comportamiento de la nave del jugador:

nave.py

```
importar pygame

nave de clase:
    """Una clase para manejar la nave."""

    def __init__(self, ai_game):
        """Inicializar el barco y establecer su posición inicial."""
        self.pantalla = ai_game.pantalla
ultravioleta  self.screen_rect = ai_game.screen.get_rect()

        # Cargue la imagen del barco y obtenga su rect.
en      self.imagen = pygame.image.load('images/ship.bmp')
        self.rect = self.imagen.get_rect()
```

```
x        # Comience cada nuevo barco en la parte inferior central de la pantalla.
         self.rect.midbottom = self.screen_rect.midbottom

y def blitme(self):
         """Dibuja el barco en su ubicación actual."""
         self.screen.blit(self.image, self.rect)
```

Pygame es eficiente porque te permite tratar todos los elementos del juego como rect ángulos *(rectángulos),* incluso si no tienen exactamente la forma de rectángulos. Tratar un elemento como un rectángulo es eficiente porque los rectángulos son formas geométricas simples. Cuando Pygame necesita averiguar si dos elementos del juego han chocado, por ejemplo, puede hacerlo más rápidamente si trata cada objeto como un rectángulo. Este enfoque generalmente funciona lo suficientemente bien como para que nadie que juegue se dé cuenta de que no estamos trabajando con la forma exacta de cada elemento del juego. Trataremos el barco y la pantalla como ángulos rectos en esta clase.

Importamos el módulo pygame antes de definir la clase. El __init__() El método Ship toma dos parámetros: la referencia propia y una referencia a la instancia actual de la clase AlienInvasion . Esto le dará a Ship acceso a todos los recursos del juego definidos en AlienInvasion. En u asignamos la pantalla a un atributo de Ship, para que podamos acceder a ella fácilmente en todos los métodos de esta clase. En v accedemos al atributo rect de la pantalla usando el método get_rect() y lo asignamos a self.screen_rect. Si lo hace, nos permite colocar el barco en la ubicación correcta en la pantalla.

Para cargar la imagen, llamamos a pygame.image.load() w y le damos la ubicación de la imagen de nuestra nave. Esta función devuelve una superficie que representa el barco, que asignamos a self.image. Cuando se carga la imagen, llamamos a get_rect() para acceder al atributo rect de la superficie del barco para que luego podamos usarlo para colocar el barco.

Cuando trabaja con un objeto rect , puede usar las coordenadas x e y de los bordes superior, inferior, izquierdo y derecho del rectángulo, así como el centro, para colocar el objeto. Puede establecer cualquiera de estos valores para establecer la posición actual del rect. Cuando esté centrando un elemento del juego, trabaje con los atributos center, centerx o centery de un rect. Cuando esté trabajando en un borde de la pantalla, trabaje con los atributos superior, inferior, izquierdo o derecho . También hay atributos que combinan estas propiedades, como midbottom, midtop, midleft y midright. Cuando esté ajustando la ubicación horizontal o vertical del rect, puede usar los atributos x e y , que son las coordenadas x e y de su esquina superior izquierda. Estos atributos le evitan tener que hacer cálculos que antes los desarrolladores de juegos tenían que hacer manualmente, y los usará con frecuencia.

En Pygame, el origen (0, 0) está en la esquina superior izquierda de la pantalla y las coordenadas aumentan a medida que avanza hacia abajo y hacia la derecha. En una pantalla de 1200 por 800, el origen está en la esquina superior izquierda y la esquina inferior derecha tiene las coordenadas (1200, 800). Estas coordenadas se refieren a la ventana del juego, no a la pantalla física.

Colocaremos la nave en la parte inferior central de la pantalla. Para hacerlo, haga que el valor de self.rect.midbottom coincida con el atributo midbottom del rect x de la pantalla. Pygame usa estos atributos rect para posicionar la imagen de la nave de modo que esté centrada horizontalmente y alineada con la parte inferior de la pantalla.

En y, definimos el método blitme() , que dibuja la imagen en la pantalla en la posición especificada por self.rect.

Dibujar el barco en la pantalla

Ahora actualicemos *alien_invasion.py* para que cree una nave y llame al método blitme() de la nave :

alien_invasion.py

```
--recorte--
desde ajustes importar Ajustes
desde barco importación Barco

clase invasión alienígena:
    """Clase general para administrar los activos y el comportamiento del juego"."""

    def __init__(uno mismo):
        --recorte--
        pygame.display.set_caption("Invasión alienígena")

en      self.barco = Barco(uno mismo)

    def run_game(self):
            --recorte--
            # Vuelva a dibujar la pantalla durante cada pasada por el bucle.
            self.screen.fill(self.settings.bg_color)
en          self.ship.blitme()

            # Hacer visible la pantalla dibujada más recientemente.
            pygame.display.flip()
--recorte--
```

Importamos Ship y luego creamos una instancia de Ship después de que se haya creado la pantalla u. La llamada a Ship() requiere un argumento, una instancia de AlienInvasion. El argumento propio aquí se refiere a la instancia actual de AlienInvasion. Este es el parámetro que le da a Ship acceso a los recursos del juego, como el objeto de pantalla . Asignamos esta instancia de Ship a self.ship.

Después de llenar el fondo, dibujamos el barco en la pantalla llamando ship.blitme(), por lo que el barco aparece en la parte superior del fondo v.

Cuando *ejecutes alien_invasion.py* ahora, deberías ver un juego vacío pantalla con el cohete sentado en la parte inferior central, como se muestra en la Figura 12-2.

Figura 12-2: Invasión alienígena con la nave en la parte inferior central de la pantalla

Refactorización: _check_events() y _update_screen() Métodos

En proyectos grandes, a menudo refactorizará el código que ha escrito antes de agregar más código. La refactorización simplifica la estructura del código que ya ha escrito, lo que facilita la creación. En esta sección, dividiremos el run_game() método, que se está haciendo largo, en dos métodos auxiliares. Un *método auxiliar* funciona dentro de una clase, pero no debe llamarse a través de una instancia. En Python, un único guión bajo inicial indica un método auxiliar.

El método _check_events()

Moveremos el código que administra los eventos a un método separado llamado _check_events(). Esto simplificará run_game() y aislará el bucle de gestión de eventos. Aislar el bucle de eventos te permite administrar los eventos por separado de otros aspectos del juego, como actualizar la pantalla.

Aquí está la clase AlienInvasion con el nuevo método _check_events() , que solo afecta el código en run_game():

alien_invasion.py

```
    def run_game(self):
        """Iniciar el bucle principal del juego."""
        mientras que es cierto:
en          self._check_events()
```

```
        # Vuelva a dibujar la pantalla durante cada pasada por el bucle.
        --recorte--

v def _check_events(auto):
        """Responder a las pulsaciones de teclas y eventos del ratón."""
        para evento en pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
```

Creamos un nuevo método _check_events() v y movemos las líneas que verifican si el jugador ha hecho clic para cerrar la ventana en este nuevo método.

Para llamar a un método desde dentro de una clase, use la notación de puntos con la variable self y el nombre del método u. Llamamos al método desde dentro del ciclo while en run_game().

El método _update_screen()

Para simplificar aún más run_game(), moveremos el código para actualizar la pantalla a un método separado llamado _update_screen():

alien_invasion.py

```
    def run_game(self):
        """Iniciar el bucle principal del juego."""
        mientras que es cierto:
            self._check_events()
            self._update_screen()

    def _check_events(auto):
        --recorte--

    def _update_screen(auto):
        """Actualice las imágenes en la pantalla y pase a la nueva pantalla."""
        self.screen.fill(self.settings.bg_color)
        self.ship.blitme()

        pygame.display.flip()
```

Movimos el código que dibuja el fondo y el barco y cambia la pantalla a _update_screen(). Ahora el cuerpo del ciclo principal en run_game() es mucho más simple. Es fácil ver que estamos buscando nuevos eventos y actualizando la pantalla en cada pasada por el bucle.

Si ya ha creado varios juegos, probablemente comenzará dividiendo su código en métodos como estos. Pero si nunca ha abordado un proyecto como este, probablemente no sabrá cómo estructurar su código. Este enfoque de escribir código que funciona y luego reestructurarlo a medida que se vuelve más complejo le da una idea de un proceso de desarrollo realista: comienza escribiendo su código de la manera más simple posible y luego lo refactoriza a medida que su proyecto se vuelve más complejo.

Ahora que hemos reestructurado el código para que sea más fácil agregarlo, ¡podemos trabajar en los aspectos dinámicos del juego!

Inténtalo tú mismo

12-1. Cielo azul: haga una ventana de Pygame con un fondo azul.

12-2. Personaje del juego: encuentre una imagen de mapa de bits de un personaje del juego que le guste o convierta una imagen en un mapa de bits. Cree una clase que dibuje al personaje en el centro de la pantalla y haga coincidir el color de fondo de la imagen con el color de fondo de la pantalla, o viceversa.

pilotando el barco

A continuación, le daremos al jugador la capacidad de mover la nave hacia la derecha y hacia la izquierda. Escribiremos un código que responda cuando el jugador presione la tecla de flecha derecha o izquierda. Primero nos concentraremos en el movimiento hacia la derecha y luego aplicaremos los mismos principios para controlar el movimiento hacia la izquierda. A medida que agreguemos este código, aprenderá cómo controlar el movimiento de las imágenes en la pantalla y cómo responder a las entradas del usuario.

Responder a una pulsación de tecla

Cada vez que el jugador presiona una tecla, esa pulsación de tecla se registra en Pygame como un evento. Cada evento es recogido por el método pygame.event.get() . Necesitamos especificar en nuestro método _check_events() qué tipo de eventos queremos que el juego verifique. Cada pulsación de tecla se registra como un evento KEYDOWN .

Cuando Pygame detecta un evento KEYDOWN , debemos verificar si la tecla que se presionó es una que desencadena una determinada acción. Por ejemplo, si el jugador presiona la tecla de flecha derecha, queremos aumentar el valor rect.x del barco para mover el barco a la derecha:

alien_invasion.py

```
def _check_events(auto):
    """Responder a las pulsaciones de teclas y eventos del ratón."""
    para evento en pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.KEYDOWN:
ultravioleta    si evento.key == pygame.K_RIGHT:
                # Mueve el barco a la derecha.
en              self.ship.rect.x += 1
```

Dentro de _check_events() agregamos un bloque elif al bucle de eventos para responder cuando Pygame detecta un evento KEYDOWN u. Verificamos si la tecla presionada, event.key, es la tecla de flecha derecha v. La tecla de flecha derecha está representada por pygame.K_RIGHT. Si se presionó la tecla de flecha derecha, movemos el barco hacia la derecha aumentando el valor de self.ship.rect.x en 1 w.

Cuando ejecute *alien_invasion.py* ahora, la nave debería moverse un píxel a la derecha cada vez que presione la tecla de flecha derecha. Eso es un comienzo, pero no es una forma eficiente de controlar la nave. Mejoremos este control permitiendo el movimiento continuo.

Permitir el movimiento continuo

Cuando el jugador mantiene presionada la tecla de flecha hacia la derecha, queremos que la nave continúe moviéndose hacia la derecha hasta que el jugador suelte la tecla. Haremos que el juego detecte un evento pygame.KEYUP para saber cuándo se suelta la tecla de flecha derecha; luego usaremos los eventos KEYDOWN y KEYUP junto con un indicador llamado moving_right para implementar el movimiento continuo.

Cuando la bandera de movimiento a la derecha es falsa , el barco estará inmóvil. Cuando el jugador presione la tecla de flecha derecha, configuraremos la bandera en Verdadero, y cuando el jugador suelte la tecla, configuraremos la bandera en Falso nuevamente.

La clase Barco controla todos los atributos del barco, por lo que le daremos un atributo llamado moviendo_derecha y un método de actualización() para verificar el estado de la bandera moviendo_derecha . El método update() cambiará la posición del barco si la bandera se establece en True. Llamaremos a este método una vez en cada pasada por el bucle while para actualizar la posición de la nave.

Estos son los cambios en Enviar:

nave.py

```
nave de clase:
    """Una clase para manejar la nave."""

    def __init__(self, ai_game):
        --recorte--
        # Comience cada nuevo barco en la parte inferior central de la pantalla.
        self.rect.midbottom = self.screen_rect.midbottom

        # Bandera de movimiento
en      self.moving_right = Falso

v def actualizar (auto):
        """Actualiza la posición del barco según la bandera de movimiento."""
        if self.moving_right:
            self.rect.x += 1

    def blitme(yo):
        --recorte--
```

Agregamos un atributo self.moving_right en el método __init__() y lo configuramos como False inicialmente u. Luego agregamos update(), que mueve el barco a la derecha si la bandera es True v. El método update() se llamará a través de una instancia de Ship, por lo que no se considera un método auxiliar.

Ahora necesitamos modificar _check_events() para que moving_right se establezca en True cuando se presiona la tecla de flecha derecha y False cuando se suelta la tecla:

alien_invasion.py

```
    def _check_events(auto):
        """Responder a las pulsaciones de teclas y eventos del ratón."""
        para evento en pygame.event.get():
            --recorte--
            elif event.type == pygame.KEYDOWN:
                si evento.key == pygame.K_RIGHT:
                    self.ship.moving_right = Verdadero
ultravioleta elif event.type == pygame.KEYUP:
```

```
            si evento.key == pygame.K_RIGHT:
                self.ship.moving_right = Falso
```

En u, modificamos cómo responde el juego cuando el jugador presiona la tecla de flecha hacia la derecha: en lugar de cambiar la posición de la nave directamente, simplemente establecemos moving_right en True. En v, agregamos un nuevo bloque elif , que responde a eventos KEYUP . Cuando el jugador suelta la tecla de flecha hacia la derecha (K_RIGHT), establecemos moving_right en False.

A continuación, modificamos el ciclo while en run_game() para que llame a la actualización del barco () método en cada paso a través del bucle:

alien_invasion.py

```
    def run_game(self):
        """Iniciar el bucle principal del juego."""
        mientras que es cierto:
            self._check_events()
            self.ship.update()
            self._update_screen()
```

La posición del barco se actualizará después de que hayamos verificado los eventos del teclado y antes de que actualicemos la pantalla. Esto permite que la posición del barco se actualice en respuesta a la entrada del jugador y garantiza que la posición actualizada se utilizará al dibujar el barco en la pantalla.

Cuando ejecuta *alien_invasion.py* y mantiene presionada la tecla de flecha derecha, el el barco debe moverse continuamente hacia la derecha hasta que suelte la tecla.

Mover tanto a la izquierda como a la derecha

Ahora que el barco puede moverse continuamente hacia la derecha, agregar movimiento hacia la izquierda es sencillo. Nuevamente, modificaremos la clase Ship y el _check método _eventos() . Aquí están los cambios relevantes a __init__() y update() en barco:

nave.py

```
    def __init__(self, ai_game):
        --recorte--
        # Banderas de movimiento
        self.moving_right = Falso
        self.moving_left = Falso

    def actualizar (auto):
        """Actualizar la posición del barco según las banderas de movimiento."""
        if self.moving_right:
            self.rect.x += 1
        si self.moving_left:
            self.rect.x -= 1
```

En __init__(), agregamos un indicador self.moving_left . En update(), usamos dos bloques if separados en lugar de un elif para permitir que el valor rect.x de la nave aumente y luego disminuya cuando ambas teclas de flecha se mantienen presionadas. Esto da como resultado que el barco se detenga. Si usamos elif para el movimiento hacia la izquierda, el

la tecla de flecha derecha siempre tendría prioridad. Hacerlo de esta manera hace que los movimientos sean más precisos al cambiar de derecha a izquierda cuando el jugador puede mantener presionadas ambas teclas momentáneamente.

Tenemos que hacer dos ajustes a _check_events():

alien_invasion.py

```
def _check_events(auto):
    """Responder a las pulsaciones de teclas y eventos del ratón."""
    para evento en pygame.event.get():
        --recorte--
        elif event.type == pygame.KEYDOWN:
            si evento.key == pygame.K_RIGHT:
                self.ship.moving_right = Verdadero
            elif evento.key == pygame.K_LEFT:
                self.ship.moving_left = Verdadero

        elif event.type == pygame.KEYUP:
            si evento.key == pygame.K_RIGHT:
                self.ship.moving_right = Falso
            elif evento.key == pygame.K_LEFT:
                self.ship.moving_left = Falso
```

Si ocurre un evento KEYDOWN para la tecla K_LEFT , establecemos moving_left en True. Si ocurre un evento KEYUP para la tecla K_LEFT , establecemos moving_left en False. Podemos usar bloques elif aquí porque cada evento está conectado a una sola clave. Si el jugador presiona ambas teclas a la vez, se detectarán dos eventos separados.

Cuando ejecute *alien_invasion.py* ahora, debería poder mover la nave continuamente hacia la derecha y hacia la izquierda. Si mantienes presionadas ambas teclas, el barco debería dejar de moverse.

A continuación, refinaremos aún más el movimiento del barco. Ajustemos la velocidad de la nave y limitemos hasta dónde puede moverse la nave para que no desaparezca por los lados de la pantalla.

Ajuste de la velocidad del barco

Actualmente, la nave se mueve un píxel por ciclo a través del bucle while , pero podemos tener un control más preciso de la velocidad de la nave agregando un atributo ship_speed a la clase Configuración . Usaremos este atributo para determinar qué tan lejos mover el barco en cada pasada por el bucle. Aquí está el nuevo atributo en *settings.py:*

configuración.py

```
Configuración de clase:
    """Una clase para almacenar todos los ajustes de Alien Invasion."""

    def __init__(uno mismo):
        --recorte--

        # Configuración de envío
        self.ship_speed = 1.5
```

Establecemos el valor inicial de ship_speed en 1.5. Cuando el barco se mueve ahora, su posición se ajusta en 1,5 píxeles en lugar de 1 píxel en cada paso por el bucle.

Usamos valores decimales para la configuración de velocidad para tener un control más preciso de la velocidad del barco cuando aumentamos el ritmo del juego más adelante. Sin embargo, los atributos rect como x almacenan solo valores enteros, por lo que debemos realizar algunas modificaciones en Ship:

nave.py

```
nave de clase:
    """Una clase para manejar la nave."""

u   def __init__(self, ai_game):
        """Inicializar el barco y establecer su posición inicial."""
        self.pantalla = ai_game.pantalla
        self.settings = ai_game.settings
        --recorte--

        # Comience cada nuevo barco en la parte inferior central de la pantalla.
        --recorte--

        # Almacena un valor decimal para la posición horizontal del barco.
en      self.x = float(self.rect.x)

        # Banderas de movimiento
        self.moving_right = Falso
        self.moving_left = Falso

    def actualizar (auto):
        """Actualizar la posición del barco según las banderas de movimiento."""
        # Actualice el valor x del barco, no el rect.
        if self.moving_right:
en          self.x += self.settings.ship_speed
        si self.moving_left:
            self.x -= self.settings.ship_speed

        # Actualizar el objeto rect de self.x.
X       self.rect.x = self.x

    def blitme(yo):
        --recorte--
```

Creamos un atributo de configuración para Ship, para que podamos usarlo en update() u. Debido a que estamos ajustando la posición del barco en fracciones de un píxel, debemos asignar la posición a una variable que pueda almacenar un valor decimal. Puede usar un valor decimal para establecer un atributo de rect, pero rect solo conservará la parte entera de ese valor. Para realizar un seguimiento preciso de la posición del barco, definimos un nuevo atributo self.x que puede contener valores decimales v. Usamos la función float() para convertir el valor de self.rect.x a un decimal y asignamos este valor a self.x.

Ahora, cuando cambiamos la posición del barco en update(), el valor de self.x se ajusta por la cantidad almacenada en settings.ship_speed w. Después de actualizar self.x , usamos el nuevo valor para actualizar self.rect.x, que controla

la posición del barco x. Solo la parte entera de self.x se almacenará en self.rect.x, pero está bien para mostrar la nave.

Ahora podemos cambiar el valor de ship_speed, y cualquier valor mayor que uno hará que el barco se mueva más rápido. Esto ayudará a que la nave responda lo suficientemente rápido como para derribar a los alienígenas, y nos permitirá cambiar el ritmo del juego a medida que el jugador avanza en el juego.

Si está utilizando macOS, es posible que observe que el barco se mueve muy lentamente, incluso con una configuración de alta velocidad. Puedes solucionar este problema ejecutando el juego en modo de pantalla completa, que implementaremos en breve.

Limitación del alcance del barco

En este punto, el barco desaparecerá de cualquier borde de la pantalla si mantienes presionada una tecla de flecha el tiempo suficiente. Corrijamos esto para que la nave deje de moverse cuando llegue al borde de la pantalla. Hacemos esto modificando el método update() en Ship:

nave.py

```
def actualizar (auto):
    """Actualizar la posición del barco según las banderas de movimiento."""
    # Actualice el valor x del barco, no el rect.
en  if self.moving_right y self.rect.right < self.screen_rect.right:
        self.x += self.settings.ship_speed
en  si self.moving_left y self.rect.left > 0:
        self.x -= self.settings.ship_speed

    # Actualizar el objeto rect de self.x.
    self.rect.x = self.x
```

Este código comprueba la posición del barco antes de cambiar el valor de self.x. El código self.rect.right devuelve la coordenada x del borde derecho del rect del barco. Si este valor es menor que el valor devuelto por self.screen
_rect.right, la nave no ha llegado al borde derecho de la pantalla u. Lo mismo ocurre con el borde izquierdo: si el valor del lado izquierdo del rect es mayor que cero, el barco no ha llegado al borde izquierdo de la pantalla v. Esto asegura que el barco está dentro de estos límites antes de ajustar el valor de yo.x.

Cuando ejecute *alien_invasion.py* ahora, la nave debería dejar de moverse en cualquier borde de la pantalla. Esto está muy bien; todo lo que hemos hecho es agregar una prueba condicional en una declaración if , ¡pero se siente como si la nave chocara contra una pared o un campo de fuerza en cualquiera de los bordes de la pantalla!

Refactorización de _check_events()

El método _check_events() aumentará en longitud a medida que continuamos desarrollando el juego, así que dividamos _check_events() en dos métodos más: uno que maneja eventos KEYDOWN y otro que maneja eventos KEYUP :

alien_invasion.py

```
def _check_events(auto):
    """Responder a las pulsaciones de teclas y eventos del ratón."""
    para evento en pygame.event.get():
```

```
            if event.type == pygame.QUIT:
                sys.exit()
            elif event.type == pygame.KEYDOWN:
                self._check_keydown_events(evento)
            elif event.type == pygame.KEYUP:
                self._check_keyup_events(evento)

    def _check_keydown_events(self, evento):
        """Responder a las pulsaciones de teclas."""
        si evento.key == pygame.K_RIGHT:
            self.ship.moving_right = Verdadero
        elif evento.key == pygame.K_LEFT:
            self.ship.moving_left = Verdadero

    def _check_keyup_events(self, evento):
        """Responder a comunicados de clave."""
        si evento.key == pygame.K_RIGHT:
            self.ship.moving_right = Falso
        elif evento.key == pygame.K_LEFT:
            self.ship.moving_left = Falso
```

Creamos dos nuevos métodos auxiliares: _check_keydown_events() y _check _keyup_events(). Cada uno necesita un parámetro propio y un parámetro de evento . Los cuerpos de estos dos métodos se copian de _check_events(), y hemos reemplazado el código anterior con llamadas a los nuevos métodos. Los _check_events()
El método es más simple ahora con esta estructura de código más limpia, lo que facilitará el desarrollo de más respuestas a la entrada del jugador.

Presionar Q para salir

Ahora que estamos respondiendo a las pulsaciones de teclas de manera eficiente, podemos agregar otra forma de salir del juego. Se vuelve tedioso hacer clic en la X en la parte superior de la ventana del juego para finalizar el juego cada vez que prueba una nueva función, por lo que agregaremos un atajo de teclado para finalizar el juego cuando el jugador presione Q:

alien_invasion.py

```
    def _check_keydown_events(self, evento):
        --recorte--
        elif evento.key == pygame.K_LEFT:
            self.ship.moving_left = Verdadero
        elif evento.key == pygame.K_q:
            sys.exit()
```

En _check_keydown_events(), agregamos un nuevo bloque que finaliza el juego cuando el jugador presiona Q. Ahora, al probar, puede presionar Q para cerrar el juego en lugar de usar el cursor para cerrar la ventana.

Ejecutar el juego en modo de pantalla completa

Pygame tiene un modo de pantalla completa que te puede gustar más que ejecutar el juego en una ventana normal. Algunos juegos se ven mejor en el modo de pantalla completa y los usuarios de macOS pueden ver un mejor rendimiento en el modo de pantalla completa.

Para ejecutar el juego en modo de pantalla completa, realice los siguientes cambios en __init__():

alien_invasion.py

```
def __init__(uno mismo):
    """Inicializa el juego y crea recursos del juego."""
    pygame.init ()
    self.settings = Ajustes()

    self.pantalla = pygame.display.set_mode((0, 0), pygame.PANTALLA COMPLETA)
ultravioleta self.settings.screen_width = self.screen.get_rect().width
    self.settings.screen_height = self.screen.get_rect().height
    pygame.display.set_caption("Invasión alienígena")
```

Al crear la superficie de la pantalla, pasamos un tamaño de (0, 0) y el parámetro pygame.FULLSCREEN u. Esto le dice a Pygame que calcule un tamaño de ventana que llene la pantalla. Debido a que no conocemos el ancho y el alto de la pantalla con anticipación, actualizamos esta configuración después de que se crea la pantalla v. Usamos los atributos de ancho y alto del rect de la pantalla para actualizar el objeto de configuración .

Si te gusta cómo se ve o se comporta el juego en modo de pantalla completa, mantén esta configuración. Si te gustó más el juego en su propia ventana, puedes volver al enfoque original donde establecimos un tamaño de pantalla específico para el juego.

Asegúrate de poder salir presionando Q antes de ejecutar el juego en modo de pantalla completa; Pygame no ofrece una forma predeterminada de salir de un juego mientras está en modo de pantalla completa.

Un resumen rápido

En la siguiente sección, agregaremos la capacidad de disparar balas, lo que implica agregar un nuevo archivo llamado *bullet.py* y hacer algunas modificaciones a algunos de los archivos que ya estamos usando. En este momento, tenemos tres archivos que contienen varias clases y métodos. Para tener claro cómo está organizado el proyecto, revisemos cada uno de estos archivos antes de agregar más funcionalidades.

alien_invasion.py

El archivo principal, *alien_invasion.py,* contiene la clase AlienInvasion . Esta clase crea una serie de atributos importantes que se utilizan a lo largo del juego: la configuración se asigna a la configuración, la superficie de visualización principal se asigna a la pantalla y también se crea una instancia de barco en este archivo. El ciclo principal del juego, un ciclo while , también se almacena en este módulo. El bucle while llama a _check_events(), ship.update() y _update_screen().

El método _check_events() detecta eventos relevantes, como presionar y soltar teclas, y procesa cada uno de estos tipos de eventos a través de los métodos _check_keydown_events() y _check_keyup_events(). Por ahora,

estos métodos gestionan el movimiento del barco. La clase AlienInvasion también contiene _update_screen(), que vuelve a dibujar la pantalla en cada pasada por el bucle principal.

El archivo *alien_invasion.py* es el único archivo que necesitas ejecutar cuando quieres jugar a *Alien Invasion.* Los otros archivos, settings.py y *ship.py* , contienen código que se importa a este archivo.

configuración.py

El archivo *settings.py* contiene la clase Settings . Esta clase solo tiene un __init__() método, que inicializa los atributos que controlan la apariencia del juego y la velocidad del barco.

nave.py

El archivo *ship.py* contiene la clase Ship . La clase Ship tiene un __init__() un método update() para gestionar la posición del barco y un método blitme() método para dibujar el barco en la pantalla. La imagen del barco se almacena en *ship.bmp,* que se encuentra en la carpeta de *imágenes* .

Inténtalo tú mismo

12-3. Documentación de Pygame: estamos lo suficientemente avanzados en el juego ahora que es posible que desee ver parte de la documentación de Pygame. La página de inicio de Pygame está en https://www.pygame.org/, y la página de inicio de la documentación está en https://www.pygame.org/docs/. Solo hojee la documentación por ahora.
No lo necesitarás para completar este proyecto, pero te ayudará si quieres modificar Alien Invasion o crear tu propio juego después.

12-4. Cohete: haz un juego que comience con un cohete en el centro de la pantalla. Permita que el jugador mueva el cohete hacia arriba, hacia abajo, hacia la izquierda o hacia la derecha con las cuatro teclas de flecha. Asegúrese de que el cohete nunca se mueva más allá de cualquier borde de la pantalla.

12-5. Teclas: Cree un archivo Pygame que cree una pantalla vacía. En el bucle de eventos, imprima el atributo event.key siempre que se detecte un evento pygame.KEYDOWN .
Ejecute el programa y presione varias teclas para ver cómo responde Pygame.

disparar balas

Ahora agreguemos la capacidad de disparar balas. Escribiremos un código que dispare una bala, que está representada por un pequeño rectángulo, cuando el jugador presione la barra espaciadora. Las balas viajarán directamente hacia arriba en la pantalla hasta que desaparezcan de la parte superior de la pantalla.

Agregar la configuración de viñetas

Al final del método __init__() , actualizaremos *settings.py* para incluir los valores que necesitaremos para una nueva clase Bullet :

configuración.py

```
def __init__(uno mismo):
    --recorte--
    # Configuración de viñetas
    self.bullet_speed = 1.0
    self.bullet_width = 3
    self.bullet_height = 15
    self.bullet_color = (60, 60, 60)
```

Estos ajustes crean viñetas de color gris oscuro con un ancho de 3 píxeles y un altura de 15 píxeles. Las balas viajarán un poco más lento que el barco.

Crear la clase de viñetas

Ahora cree un archivo *bullet.py* para almacenar nuestra clase Bullet . Aquí está la primera parte de *bullet.py:*

bala.py

```
importar pygame
de pygame.sprite importar Sprite

Clase Bala (Sprite):
    """Una clase para manejar las balas disparadas desde el barco"""

    def __init__(self, ai_game):
        """Crea un objeto de bala en la posición actual del barco".""
        super().__init__()
        self.pantalla = ai_game.pantalla
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color

        # Cree un rectángulo de bala en (0, 0) y luego establezca la posición correcta.
en      self.rect = pygame.Rect(0, 0, self.settings.bullet_width,
            self.settings.bullet_height)
en      self.rect.midtop = ai_game.ship.rect.midtop

        # Almacena la posición de la viñeta como un valor decimal.
en      self.y = float(self.rect.y)
```

La clase Bullet hereda de Sprite, que importamos del pygame Módulo .sprite . Cuando usa sprites, puede agrupar elementos relacionados en su juego y actuar en todos los elementos agrupados a la vez. Para crear una instancia de viñeta, __init__() necesita la instancia actual de AlienInvasion, y llamamos a super() para heredar correctamente de Sprite. También establecemos atributos para la pantalla y los objetos de configuración, y para el color de la viñeta.

En u, creamos el atributo rect de la viñeta. La viñeta no se basa en una imagen, así que tenemos que construir un rect desde cero usando la clase pygame.Rect() . Esta clase requiere las coordenadas x e y de la esquina superior izquierda de la

rect, y el ancho y alto del rect. Inicializamos el rect en (0, 0), pero lo moveremos a la ubicación correcta en la siguiente línea, porque la posición de la bala depende de la posición del barco. Obtenemos el ancho y el alto de la viñeta de los valores almacenados en self.settings.

En v, configuramos el atributo midtop de la bala para que coincida con el atributo midtop del barco . Esto hará que la bala emerja de la parte superior de la nave, haciendo que parezca que la bala fue disparada desde la nave. Almacenamos un valor decimal para la coordenada y de la bala para que podamos hacer ajustes finos a la velocidad w de la bala.

Aquí está la segunda parte de *bullet.py,* update() y draw_bullet():

bala.py

```
    def actualizar (auto):
        """Mueve la bala hacia arriba en la pantalla."""
        # Actualizar la posición decimal de la viñeta.
en      self.y -= self.settings.bullet_speed
        # Actualizar la posición del rect.
en      self.rect.y = self.y

    def dibujar_bullet(uno mismo):
        """Dibuja la viñeta hacia la pantalla."""
en      pygame.draw.rect(self.screen, self.color, self.rect)
```

El método update() administra la posición de la viñeta. Cuando se dispara una bala, se mueve hacia arriba en la pantalla, lo que corresponde a un valor de coordenada y decreciente. Para actualizar la posición, restamos la cantidad almacenada en la configuración .bullet_speed de self.y u. Luego usamos el valor de self.y para establecer el valor de self.rect.y v.

La configuración bullet_speed nos permite aumentar la velocidad de las balas a medida que avanza el juego o según sea necesario para refinar el comportamiento del juego. Una vez que se dispara una bala, nunca cambiamos el valor de su coordenada x, por lo que viajará verticalmente en línea recta incluso si el barco se mueve.

Cuando queremos dibujar una viñeta, llamamos a draw_bullet(). El dibujo.rect() La función llena la parte de la pantalla definida por el rect de la viñeta con el color almacenado en self.color w.

Almacenamiento de viñetas en un grupo

Ahora que tenemos una clase Bullet y las configuraciones necesarias definidas, podemos escribir código para disparar una bala cada vez que el jugador presione la barra espaciadora. Crearemos un grupo en AlienInvasion para almacenar todas las balas vivas para que podamos administrar las balas que ya se han disparado. Este grupo será una instancia de la clase pygame.sprite.Group , que se comporta como una lista con algunas funciones adicionales que son útiles al crear juegos. Usaremos este grupo para dibujar viñetas en la pantalla en cada pasada por el bucle principal y para actualizar la posición de cada viñeta.

Crearemos el grupo en __init__():

alien_invasion.py

```
    def __init__(uno mismo):
        --recorte--
        self.barco = Barco(uno mismo)
        self.viñetas = pygame.sprite.Group()
```

Luego necesitamos actualizar la posición de las viñetas en cada pasada a través del ciclo while :

alien_invasion.py

```
    def run_game(self):
        """Iniciar el bucle principal del juego."""
        mientras que es cierto:
            self._check_events()
            self.ship.update()
en          self.bullets.update()
            self._update_screen()
```

Cuando llamas a update() en un grupo u, el grupo automáticamente llama a update() para cada sprite en el grupo. La línea self.bullets.update() llama a bullet.update() para cada viñeta que colocamos en el grupo de viñetas.

disparando balas

En AlienInvasion, necesitamos modificar _check_keydown_events() para disparar una bala cuando el jugador presiona la barra espaciadora. No necesitamos cambiar _check_keyup _events() porque no pasa nada cuando se suelta la barra espaciadora. También necesitamos modificar _update_screen() para asegurarnos de que cada viñeta se dibuje en la pantalla antes de llamar a flip().

Sabemos que habrá un poco de trabajo por hacer cuando disparemos una bala, así que escribamos un nuevo método, _fire_bullet(), para manejar este trabajo:

alien_invasion.py

```
--recorte--
desde barco importación Barco
u de viñeta importar viñeta

clase invasión alienígena:
    --recorte--
    def _check_keydown_events(self, evento):
        --recorte--
        elif evento.key == pygame.K_q:
            sys.exit()
en      elif evento.key == pygame.K_SPACE:
            self._fire_bullet()

    def _check_keyup_events(self, evento):
        --recorte--

    def _fire_bullet(auto):
        """Cree una nueva viñeta y agréguela al grupo de viñetas."""
        new_bullet = Bullet(self)
wx      self.viñetas.añadir(nueva_viñeta)

    def _update_screen(auto):
        """Actualice las imágenes en la pantalla y pase a la nueva pantalla."""
        self.screen.fill(self.settings.bg_color)
        self.ship.blitme()
y       para viñeta en self.bullets.sprites():
            bala.draw_bullet()
```

```
        pygame.display.flip()
--recorte--
```

Primero, importamos Bullet u. Luego llamamos a _fire_bullet() cuando se presiona la barra espaciadora v. En _fire_bullet(), creamos una instancia de Bullet y la llamamos new_bullet w. Luego lo agregamos a las viñetas de grupo usando el método add() x. El método add() es similar a append(), pero es un método escrito específicamente para grupos de Pygame.

El método bullets.sprites() devuelve una lista de todos los sprites en el grupo de viñetas. Para dibujar todas las balas disparadas en la pantalla, recorremos los sprites en balas y llamamos a draw_bullet() en cada uno y.

Cuando *ejecutes alien_invasion.py* ahora, deberías poder mover la nave hacia la derecha y hacia la izquierda y disparar tantas balas como quieras. Las balas suben por la pantalla y desaparecen cuando llegan a la parte superior, como se muestra en la figura 12-3. Puede modificar el tamaño, el color y la velocidad de las viñetas en *settings.py.*

Figura 12-3: La nave después de disparar una serie de balas

Eliminación de viñetas antiguas

Por el momento, las viñetas desaparecen cuando llegan a la parte superior, pero solo porque Pygame no puede dibujarlas sobre la parte superior de la pantalla. Las balas en realidad siguen existiendo; sus valores de coordenadas y se vuelven cada vez más negativos. Esto es un problema, porque continúan consumiendo memoria y potencia de procesamiento.

Tenemos que deshacernos de estas viejas balas, o el juego se ralentizará de haciendo tanto trabajo innecesario. Para hacer esto, necesitamos detectar cuándo el valor inferior del rect de una viñeta tiene un valor de 0, lo que indica que la viñeta ha pasado por la parte superior de la pantalla:

alien_invasion.py

```
def run_game(self):
    """Iniciar el bucle principal del juego."""
    mientras que es cierto:
        self._check_events()
        self.ship.update()
        self.bullets.update()

        # Deshazte de las balas que han desaparecido.
en      para viñeta en self.bullets.copy():
            si viñeta.rect.bottom <= 0:
volvo               self.bullets.remove(viñeta)
X       imprimir (len (auto.viñetas))

        self._update_screen()
```

Cuando usa un bucle for con una lista (o un grupo en Pygame), Python espera que la lista mantenga la misma longitud mientras el bucle se esté ejecutando. Debido a que no podemos eliminar elementos de una lista o grupo dentro de un ciclo for , tenemos que recorrer una copia del grupo. Usamos el método copy() para configurar el bucle for u, que nos permite modificar viñetas dentro del bucle. Verificamos cada viñeta para ver si ha desaparecido de la parte superior de la pantalla en v. Si es así, la eliminamos de las viñetas w. En x, insertamos una llamada print() para mostrar cuántas viñetas existen actualmente en el juego y verificamos que se eliminen cuando lleguen a la parte superior de la pantalla.

Si este código funciona correctamente, podemos observar la salida de la terminal mientras disparamos balas y ver que el número de balas se reduce a cero después de que cada serie de balas haya pasado por la parte superior de la pantalla. Después de ejecutar el juego y verificar que las viñetas se eliminen correctamente, elimine la llamada print() . Si lo deja, el juego se ralentizará significativamente porque lleva más tiempo escribir la salida en el terminal que dibujar gráficos en la ventana del juego.

Limitación del número de balas

Muchos juegos de disparos limitan la cantidad de balas que un jugador puede tener en la pantalla al mismo tiempo; hacerlo anima a los jugadores a disparar con precisión. Haremos lo mismo en *Alien Invasion.*

Primero, almacene la cantidad de viñetas permitidas en *settings.py:*

configuración.py

```
# Configuración de viñetas
--recorte--
self.bullet_color = (60, 60, 60)
self.bullets_permitido = 3
```

Esto limita al jugador a tres balas a la vez. Usaremos esta configuración en AlienInvasion para comprobar cuántas viñetas existen antes de crear una nueva viñeta en _fire_bullet():

alien_invasion.py

```
def _fire_bullet(auto):
    """Cree una nueva viñeta y agréguela al grupo de viñetas."""
    si len(self.bullets) < self.settings.bullets_allowed:
        new_bullet = Bullet(self)
        self.viñetas.añadir(nueva_viñeta)
```

Cuando el jugador presiona la barra espaciadora, verificamos la longitud de las balas. Si len(self.bullets) es menor que tres, creamos una nueva viñeta. Pero si ya hay tres balas activas, no pasa nada cuando se presiona la barra espaciadora.

Cuando ejecutes el juego ahora, deberías poder disparar balas solo en grupos de tres.

Creando el método _update_bullets()

Queremos mantener la clase AlienInvasion razonablemente bien organizada, así que ahora que hemos escrito y verificado el código de administración de viñetas, podemos moverlo a un método separado. Crearemos un nuevo método llamado _update_bullets() y agréguelo justo antes de _update_screen():

alien_invasion.py

```
def _update_bullets(auto):
    """Actualizar la posición de las viñetas y deshacerse de las viñetas viejas."""
    # Actualizar posiciones de viñetas.
    self.bullets.update()

    # Deshazte de las balas que han desaparecido.
    para viñeta en self.bullets.copy():
        si viñeta.rect.bottom <= 0:
            self.bullets.remove(viñeta)
```

El código para _update_bullets() se corta y pega desde run_game(); todo lo que hemos hecho aquí es aclarar los comentarios.

El ciclo while en run_game() parece simple nuevamente:

alien_invasion.py

```
    mientras que es cierto:
        self._check_events()
        self.ship.update()
        self._update_bullets()
        self._update_screen()
```

Ahora nuestro bucle principal contiene solo un código mínimo, por lo que podemos leer rápidamente los nombres de los métodos y comprender lo que sucede en el juego. El ciclo principal verifica la entrada del jugador y luego actualiza la posición del barco y las balas que se han disparado. Luego usamos las posiciones actualizadas para dibujar una nueva pantalla.

Ejecute *alien_invasion.py* una vez más y asegúrese de que todavía puede disparar balas sin errores.

Inténtalo tú mismo

12-6. Sideways Shooter: escribe un juego que coloque un barco en el lado izquierdo de la pantalla y permita al jugador mover el barco hacia arriba y hacia abajo. Haz que la nave dispare una bala que atraviese la pantalla cuando el jugador presione la barra espaciadora. Asegúrese de que las viñetas se eliminen una vez que desaparezcan de la pantalla.

Resumen

En este capítulo, aprendiste a hacer un plan para un juego y aprendiste la estructura básica de un juego escrito en Pygame. Aprendió a establecer un color de fondo y almacenar configuraciones en una clase separada donde puede ajustarlas más fácilmente. Viste cómo dibujar una imagen en la pantalla y darle al jugador control sobre el movimiento de los elementos del juego. Ha creado elementos que se mueven solos, como balas que vuelan por una pantalla, y ha eliminado objetos que ya no son necesarios. También aprendió a refactorizar el código en un proyecto de manera regular para facilitar el desarrollo continuo.

En el Capítulo 13, agregaremos extraterrestres a *Alien Invasion.* Al final del capítulo, podrás derribar a los alienígenas, ¡con suerte antes de que lleguen a tu nave!



13

¡Alienígenas!

En este capítulo, agregaremos extraterrestres a *Alien Invasion.* Agregaremos un alienígena cerca de la parte superior de la pantalla y luego generaremos una flota completa de alienígenas. Haremos que la flota avance hacia los lados y hacia abajo, y nos desharemos de cualquier extraterrestre alcanzado por una bala. Finalmente, limitaremos la cantidad de barcos que tiene un jugador y finalizaremos el juego cuando el jugador se quede sin barcos.

A medida que avance en este capítulo, aprenderá más sobre Pygame y sobre cómo administrar un proyecto grande. También aprenderá a detectar colisiones entre objetos del juego, como balas y extraterrestres. Detectar colisiones te ayuda a definir interacciones entre elementos en tus juegos: por ejemplo, puedes

confinar a un personaje dentro de las paredes de un laberinto o pasar una pelota entre dos personajes. Continuaremos trabajando a partir de un plan que revisamos ocasionalmente para mantener el enfoque de nuestras sesiones de escritura de código.

Antes de comenzar a escribir código nuevo para agregar una flota de extraterrestres a la pantalla, veamos el proyecto y actualicemos nuestro plan.

Revisión del proyecto

Cuando comienza una nueva fase de desarrollo en un proyecto grande, siempre es una buena idea revisar su plan y aclarar lo que desea lograr con el código que está a punto de escribir. En este capítulo, haremos lo siguiente:

- Examinar nuestro código y determinar si necesitamos refactorizar antes de implementar mencionando nuevas características.
- Agregue un único extraterrestre en la esquina superior izquierda de la pantalla con el espacio adecuado alrededor.
- Utilice el espacio alrededor del primer extraterrestre y el tamaño total de la pantalla para determinar cuántos extraterrestres pueden caber en la pantalla. Escribiremos un bucle para crear extraterrestres para llenar la parte superior de la pantalla.
- Haz que la flota se mueva hacia los lados y hacia abajo hasta que toda la flota sea derribada, un alienígena golpee la nave o un alienígena llegue al suelo. Si toda la flota es derribada, crearemos una nueva flota. Si un extraterrestre golpea la nave o el suelo, destruiremos la nave y crearemos una nueva flota.
- Limita la cantidad de barcos que el jugador puede usar y finaliza el juego cuando el jugador haya usado la cantidad asignada de barcos.

Refinaremos este plan a medida que implementemos características, pero esto es suficiente para empezar.

También debe revisar su código existente cuando comience a trabajar en una nueva serie de funciones en un proyecto. Debido a que cada nueva fase generalmente hace que un proyecto sea más complejo, es mejor limpiar cualquier código desordenado o ineficiente. Hemos estado refactorizando sobre la marcha, por lo que no hay ningún código que necesitemos refactorizar en este punto.

Creando el primer alienígena

Colocar un extraterrestre en la pantalla es como colocar un barco en la pantalla. El comportamiento de cada alienígena está controlado por una clase llamada Alien, que estructuraremos como la clase Ship . Continuaremos usando imágenes de mapa de bits para simplificar. Puede encontrar su propia imagen para un extraterrestre o usar la que se muestra en la Figura 13-1, que está disponible en los recursos del libro en *https:// nostarch.com/ pythoncrashcourse2e/.*
Esta imagen tiene un fondo gris, que coincide con el color de fondo de la pantalla. Asegúrese de guardar el archivo de imagen que elija en la carpeta de *imágenes .*

Figura 13-1: El alienígena que usaremos para construir la flota

Creando la clase alienígena

Ahora escribiremos la clase Alien y la guardaremos como *alien.py:*

alien.py

```
importar pygame
de pygame.sprite importar Sprite

clase Alien (Sprite):
    """Una clase para representar a un solo alienígena en la flota"."""

    def __init__(self, ai_game):
        """Inicializar el alienígena y establecer su posición inicial."""
        super().__init__()
        self.pantalla = ai_game.pantalla

        # Cargue la imagen alienígena y establezca su atributo rect.
        self.imagen = pygame.image.load('images/alien.bmp')
        self.rect = self.imagen.get_rect()

        # Comience cada nuevo alienígena cerca de la parte superior izquierda de la pantalla.
en      self.rect.x = self.rect.ancho
        self.rect.y = self.rect.height

        # Almacena la posición horizontal exacta del alienígena.
en      self.x = float(self.rect.x)
```

La mayor parte de esta clase es como la clase Barco , excepto por la ubicación de los alienígenas. en la pantalla. Inicialmente colocamos a cada extraterrestre cerca de la esquina superior izquierda de la pantalla; agregamos un espacio a la izquierda que es igual al ancho del extraterrestre y un espacio arriba igual a su altura u para que sea fácil de ver. somos principalmente

preocupado por la velocidad horizontal de los alienígenas, por lo que rastrearemos la posición horizontal de cada alienígena con precisión v.

Esta clase Alien no necesita un método para dibujarla en la pantalla; en su lugar, usaremos un método de grupo de Pygame que dibuja automáticamente todos los elementos de un grupo en la pantalla.

Crear una instancia del alienígena

Queremos crear una instancia de Alien para que podamos ver el primer alienígena en la pantalla. Como es parte de nuestro trabajo de configuración, agregaremos el código para esta instancia al final del método __init__() en AlienInvasion. Eventualmente, crearemos una flota completa de alienígenas, lo que será bastante trabajo, por lo que crearemos un nuevo método auxiliar llamado _create_fleet().

El orden de los métodos en una clase no importa, siempre que haya cierta coherencia en la forma en que se colocan. Colocaré _create_fleet() justo antes del método _update_screen() , pero funcionará en cualquier parte de AlienInvasion . Primero, importaremos la clase Alien .

Aquí están las declaraciones de importación actualizadas para *alien_invasion.py:*

alien_invasion.py

```
--recorte--
de importación de viñetas Viñeta
de importación alienígena Alien
```

Y aquí está el método __init__() actualizado :

alien_invasion.py

```
def __init__(uno mismo):
    --recorte--
    self.barco = Barco(uno mismo)
    self.viñetas = pygame.sprite.Group()
    self.aliens = pygame.sprite.Group()

    self._create_fleet()
```

Creamos un grupo para albergar la flota de alienígenas y lo llamamos _create_fleet(), que estamos a punto de escribir.

Aquí está el nuevo método _create_fleet() :

alien_invasion.py

```
def _create_fleet(auto):
    """Crear la flota de alienígenas"."""
    # Haz un extraterrestre.
    extranjero = extranjero (yo)
    self.aliens.add(alienígena)
```

En este método, creamos una instancia de Alien y luego la agregamos al grupo que albergará la flota. El extraterrestre se colocará en el área superior izquierda predeterminada de la pantalla, lo cual es perfecto para el primer extraterrestre.

Para que aparezca el extraterrestre, necesitamos llamar al método draw() del grupo en _update_screen():

alien_invasion.py

```
def _update_screen(auto):
    --recorte--
    para viñeta en self.bullets.sprites():
        bala.draw_bullet()
    self.aliens.draw(self.screen)

    pygame.display.flip()
```

Cuando llamas a draw() en un grupo, Pygame dibuja cada elemento en el grupo en la posición definida por su atributo rect . El método draw() requiere un argumento: una superficie sobre la que dibujar los elementos del grupo. La figura 13-2 muestra el primer extraterrestre en la pantalla.

Figura 13-2: Aparece el primer extraterrestre.

Ahora que el primer extraterrestre aparece correctamente, escribiremos el código para dibujar toda una flota.

Construyendo la flota alienígena

Para dibujar una flota, necesitamos averiguar cuántos alienígenas pueden caber en la pantalla y cuántas filas de alienígenas pueden caber en la pantalla. Primero calcularemos el espacio horizontal entre los extraterrestres y crearemos una fila; luego determinaremos el espacio vertical y crearemos una flota completa.

Determinar cuántos alienígenas caben en una fila

Para averiguar cuántos extraterrestres caben en una fila, veamos cuánto espacio horizontal tenemos. El ancho de la pantalla se almacena en settings.screen_width, pero necesitamos un margen vacío a cada lado de la pantalla. Haremos que este margen sea del ancho de un extraterrestre. Debido a que tenemos dos márgenes, el espacio disponible para los extraterrestres es el ancho de la pantalla menos dos anchos extraterrestres:

```
available_space_x = settings.screen_width – (2 * alien_width)
```

También necesitamos establecer el espacio entre alienígenas; lo haremos de un ancho alienígena. El espacio necesario para mostrar un extraterrestre es el doble de su ancho: un ancho para el extraterrestre y un ancho para el espacio vacío a su derecha. Para encontrar el número de extraterrestres que caben en la pantalla, dividimos el espacio disponible por dos veces el ancho de un extraterrestre. Usamos la *división de piso* (//), que divide dos números y descarta cualquier resto, por lo que obtendremos un número entero de alienígenas:

```
numero_alienígenas_x = espacio_disponible_x // (2 * ancho_alienígena)
```

Usaremos estos cálculos cuando creemos la flota.

Un gran aspecto de los cálculos en la programación es que no tienes que estar seguro tus fórmulas son correctas cuando las escribes por primera vez. Puedes probarlos y ver si funcionan. En el peor de los casos, tendrás una pantalla abarrotada de extraterrestres o con muy pocos extraterrestres. Luego puede revisar sus cálculos en función de lo que ve en la pantalla.

Crear una fila de alienígenas

Estamos listos para generar una fila completa de alienígenas. Debido a que nuestro código para hacer un solo extraterrestre funciona, reescribiremos _create_fleet() para hacer una fila completa de extraterrestres:

alien_invasion.py

```
    def _create_fleet(auto):
        """Crear la flota de alienígenas"."""
        # Crea un extraterrestre y encuentra el número de extraterrestres seguidos.
        # El espacio entre cada alienígena es igual al ancho de un alienígena.
        extranjero = extranjero (yo)
        alien_width = alien.rect.width
ultravioleta  available_space_x = self.settings.screen_width - (2 * alien_width)
        numero_alienígenas_x = espacio_disponible_x // (2 * ancho_alienígena)

        # Crea la primera fila de alienígenas.
X       para alien_number en el rango (number_aliens_x):
            # Crea un extraterrestre y colócalo en la fila.
            extranjero = extranjero (yo)
y           alien.x = alien_width + 2 * alien_width * alien_number
            extranjero.rect.x = extranjero.x
            self.aliens.add(alienígena)
```

Ya hemos pensado en la mayor parte de este código. Necesitamos saber el ancho y alto de alienígenas para colocar alienígenas, por lo que creamos un alienígena en u antes de realizar los cálculos. Este alienígena no formará parte de la flota, así que no lo agregues al grupo de alienígenas. En v, obtenemos el ancho del alien de su atributo rect y almacenamos este valor en alien_width para que no tengamos que seguir trabajando con el atributo rect . En w calculamos el espacio horizontal disponible para los extraterrestres y el número de extraterrestres que pueden caber en ese espacio.

A continuación, configuramos un ciclo que cuenta desde 0 hasta el número de extraterrestres que necesitamos para hacer x. En el cuerpo principal del ciclo, creamos un nuevo alienígena y luego establecemos su valor de coordenada x para colocarlo en la fila y. Cada extraterrestre es empujado hacia la derecha un ancho extraterrestre desde el margen izquierdo. A continuación, multiplicamos el ancho del alienígena por 2 para tener en cuenta el espacio que ocupa cada alienígena, incluido el espacio vacío a su derecha, y multiplicamos esta cantidad por la posición del alienígena en la fila. Usamos el atributo x del alien para establecer la posición de su rect. Luego agregamos cada nuevo alienígena al grupo de alienígenas.

Cuando ejecutes *Alien Invasion* ahora, deberías ver la primera fila de alienígenas aparecen, como en la figura 13-3.

Figura 13-3: La primera fila de extraterrestres

La primera fila está desplazada hacia la izquierda, lo que en realidad es bueno para el juego. La razón es que queremos que la flota se mueva hacia la derecha hasta que toque el borde de la pantalla, luego baje un poco, luego se mueva hacia la izquierda y así sucesivamente. Al igual que el juego clásico *Space Invaders,* este movimiento es más interesante que hacer que la flota caiga hacia abajo. Continuaremos este movimiento hasta que todos los alienígenas sean derribados o hasta que un alienígena golpee la nave o la parte inferior de la pantalla.

Según el ancho de pantalla que haya elegido, la alineación de la primera fila de alienígenas puede verse ligeramente diferente en su sistema.

Refactorización de _create_fleet()

Si el código que hemos escrito hasta ahora fuera todo lo que necesitamos para crear una flota, probablemente dejaríamos _create_fleet() como está. Pero tenemos más trabajo por hacer, así que vamos a limpiar un poco el método. Agregaremos un nuevo método auxiliar, _create_alien(), y lo llamaremos desde _create_fleet():

alien_invasion.py

```
    def _create_fleet(auto):
        --recorte--
        # Crea la primera fila de alienígenas.
        para alien_number en el rango (number_aliens_x):
            self._create_alien(número_alienígena)

    def _create_alien(self, alien_number):
        """Crea un extraterrestre y colócalo en la fila."""
        extranjero = extranjero (yo)
        alien_width = alien.rect.width
        alien.x = alien_width + 2 * alien_width * alien_number
        extranjero.rect.x = extranjero.x
        self.aliens.add(alienígena)
```

El método _create_alien() requiere un parámetro además del propio: necesita el número extranjero que se está creando actualmente. Usamos el mismo cuerpo que hicimos para _create_fleet() excepto que obtenemos el ancho de un extraterrestre dentro del método en lugar de pasarlo como argumento. Esta refactorización facilitará la adición de nuevas filas y la creación de una flota completa.

Adición de filas

Para terminar la flota, determinaremos la cantidad de filas que caben en la pantalla y luego repetiremos el ciclo para crear los alienígenas en una fila hasta que tengamos la cantidad correcta de filas. Para determinar el número de filas, encontramos el espacio vertical disponible restando la altura alienígena desde la parte superior, la altura del barco desde la parte inferior y dos alturas alienígenas desde la parte inferior de la pantalla:

```
available_space_y = settings.screen_height – (3 * alien_height) – ship_height
```

El resultado creará un espacio vacío sobre la nave, por lo que el jugador tiene algo de tiempo para comenzar a disparar a los alienígenas al comienzo de cada nivel.

Cada fila necesita un espacio vacío debajo, que haremos igual a la altura de un extraterrestre. Para encontrar el número de filas, dividimos el espacio disponible por dos veces la altura de un extraterrestre. Usamos la división de piso porque solo podemos hacer un número entero de filas. (Nuevamente, si estos cálculos están mal, lo veremos de inmediato y ajustaremos nuestro enfoque hasta que tengamos un espacio razonable).

```
number_rows = available_height_y // (2 * alien_height)
```

Ahora que sabemos cuántas filas caben en una flota, podemos repetir el código para crear una fila:

alien_invasion.py

```
    def _create_fleet(auto):
        --recorte--
        extranjero = extranjero (yo)
en      alien_width, alien_height = alien.rect.size
        available_space_x = self.settings.screen_width - (2 * alien_width)
        numero_alienígenas_x = espacio_disponible_x // (2 * ancho_alienígena)

        # Determine la cantidad de filas de alienígenas que caben en la pantalla.
        barco_altura = self.barco.rect.altura
en      available_space_y = (self.settings.screen_height -
                                (3 * alien_height) - ship_height)
        number_rows = available_space_y // (2 * alien_height)

        # Crea la flota completa de alienígenas.
en      para número_fila en el rango (número_filas):
            para alien_number en el rango (number_aliens_x):
                self._create_alien(número_alienígena, número_fila)

    def _create_alien(self, alien_number, row_number):
        """Crea un extraterrestre y colócalo en la fila."""
        extranjero = extranjero (yo)
        alien_width, alien_height = alien.rect.size
        alien.x = alien_width + 2 * alien_width * alien_number
        extranjero.rect.x = extranjero.x
X       alien.rect.y = alien.rect.height + 2 * alien.rect.height * fila_número
        self.aliens.add(alienígena)
```

Necesitamos el ancho y el alto de un alien, por lo que en u usamos el atributo size, que contiene una tupla con el ancho y el alto de un objeto rect . Para calcular el número de filas que caben en la pantalla, escribimos nuestro disponible
Cálculo de _space_y justo después del cálculo de available_space_x v. El cálculo está entre paréntesis para que el resultado se pueda dividir en dos líneas, lo que resulta en líneas de 79 caracteres o menos, como se recomienda.

Para crear varias filas, usamos dos bucles anidados: uno exterior y otro interior w. El bucle interior crea los alienígenas en una fila. El bucle exterior cuenta desde 0 hasta el número de filas que queremos; Python usa el código para hacer una sola fila y lo repite número_filas veces.

Para anidar los bucles, escriba el nuevo bucle for y sangre el código que desea repetir. (La mayoría de los editores de texto facilitan la sangría y la eliminación de la sangría de bloques de código, pero para obtener ayuda, consulte el Apéndice B). Ahora, cuando llamamos a _create_alien(), incluimos un argumento para el número de fila para que cada fila se pueda colocar más abajo en la pantalla.

La definición de _create_alien() necesita un parámetro para contener el número de fila. Dentro de _create_alien(), cambiamos el valor de la coordenada y de un extraterrestre cuando no está en la primera fila x comenzando con la altura de un extraterrestre para crear un espacio vacío en la parte superior de la pantalla. Cada fila comienza dos alturas alienígenas debajo

la fila anterior, entonces multiplicamos la altura del alienígena por dos y luego por el número de la fila. El número de la primera fila es 0, por lo que la ubicación vertical de la primera fila no cambia. Todas las filas subsiguientes se colocan más abajo en la pantalla.

Cuando ejecutes el juego ahora, deberías ver una flota completa de alienígenas, como se muestra en la Figura 13-4.

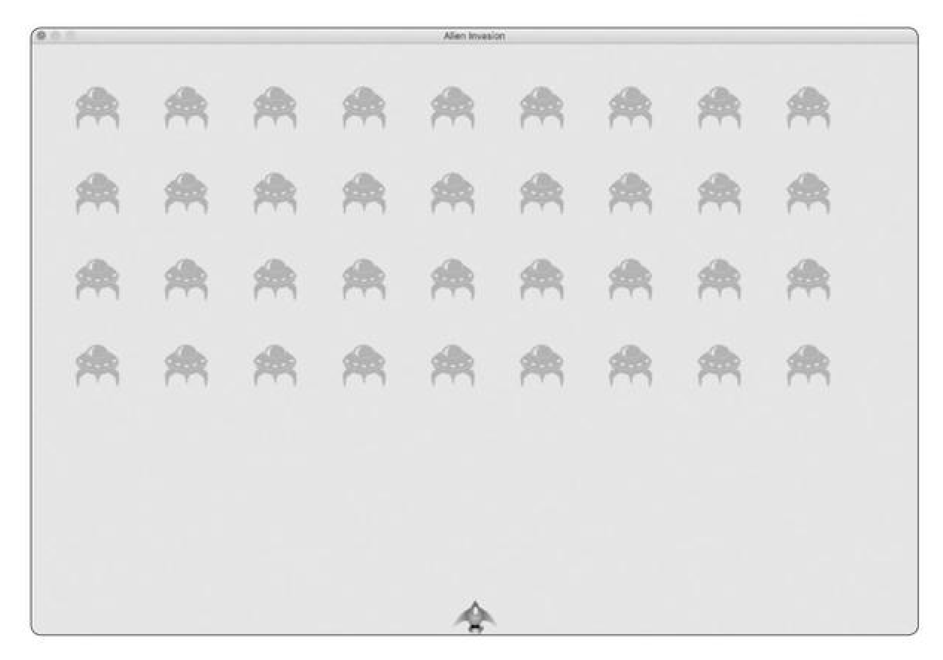


Figura 13-4: Aparece la flota completa.

¡En la siguiente sección, haremos que la flota se mueva!

Inténtalo tú mismo

13-1. Estrellas: Encuentra una imagen de una estrella. Haz que aparezca una cuadrícula de estrellas en la pantalla.

13-2. Mejores estrellas: puedes hacer un patrón de estrellas más realista introduciendo la aleatoriedad cuando colocas cada estrella. Recuerda que puedes obtener un número aleatorio como este:

```
de randint de importación aleatoria
numero_aleatorio = randint (-10, 10)
```

Este código devuelve un número entero aleatorio entre ÿ10 y 10. Usando su código en el ejercicio 13-1, ajuste la posición de cada estrella en una cantidad aleatoria.

Haciendo el movimiento de la flota

Ahora hagamos que la flota de alienígenas se mueva hacia la derecha a través de la pantalla hasta que toque el borde, y luego hagamos que suelte una cantidad determinada y se mueva en la otra dirección. Continuaremos este movimiento hasta que todos los alienígenas hayan sido derribados, uno choque con la nave o uno llegue al final de la pantalla. Comencemos haciendo que la flota se mueva hacia la derecha.

Moviendo a los alienígenas a la derecha

Para mover a los alienígenas, usaremos un método update() en *alien.py,* que llamaremos para cada alienígena en el grupo de alienígenas. Primero, agregue una configuración para controlar la velocidad de cada alienígena:

configuración.py

```
def __init__(uno mismo):
    --recorte--
    # Configuración alienígena
    self.alien_speed = 1.0
```

Luego use esta configuración para implementar la actualización ():

alien.py

```
def __init__(self, ai_game):
    """Inicializar el alienígena y establecer su posición inicial."""
    super().__init__()
    self.pantalla = ai_game.pantalla
    self.settings = ai_game.settings
    --recorte--

def actualizar (auto):
    """Mueve al alienígena a la derecha".""
    self.x += self.settings.alien_speed
ultravioleta    self.rect.x = self.x
```

Creamos un parámetro de configuración en __init__() para que podamos acceder a la velocidad del alienígena en update(). Cada vez que actualizamos la posición de un extraterrestre, lo movemos hacia la derecha en la cantidad almacenada en alien_speed. Hacemos un seguimiento de la posición exacta del extraterrestre con el atributo self.x , que puede contener valores decimales u. Luego usamos el valor de self.x para actualizar la posición del rect v del alienígena.

En el ciclo while principal , ya tenemos llamadas para actualizar las posiciones de la nave y las balas. Ahora agregaremos una llamada para actualizar también la posición de cada extraterrestre:

alien_invasion.py

```
mientras que es cierto:
    self._check_events()
    self.ship.update()
    self._update_bullets()
    self._update_aliens()
    self._update_screen()
```

Estamos a punto de escribir código para gestionar el movimiento de la flota, así que creamos un nuevo método llamado _update_aliens(). Configuramos las posiciones de los alienígenas para que se actualicen después de que se hayan actualizado las balas, porque pronto comprobaremos si alguna bala alcanza a algún alienígena.

La ubicación de este método en el módulo no es crítica. Pero para mantener el código organizado, lo colocaré justo después de _update_bullets() para que coincida con el orden de las llamadas a métodos en el ciclo while . Aquí está la primera versión de _update_aliens():

alien_invasion.py

```
def _update_aliens(auto):
    """Actualizar las posiciones de todos los alienígenas en la flota."""
    self.aliens.update()
```

Usamos el método update() en el grupo de alienígenas , que llama al método update() de cada alienígena . Cuando ejecute *Alien Invasion* ahora, debería ver que la flota se mueve hacia la derecha y desaparece del costado de la pantalla.

Creación de ajustes para la dirección de la flota

Ahora crearemos la configuración que hará que la flota se mueva hacia abajo en la pantalla y hacia la izquierda cuando toque el borde derecho de la pantalla. He aquí cómo implementar este comportamiento:

configuración.py

```
    # Configuración alienígena
    self.alien_speed = 1.0
    self.fleet_drop_speed = 10
    # Fleet_direction de 1 representa la derecha; -1 representa a la izquierda.
    self.fleet_direction = 1
```

La configuración Fleet_drop_speed controla la rapidez con la que la flota baja por la pantalla cada vez que un alienígena llega a cualquiera de los bordes. Es útil separar esta velocidad de la velocidad horizontal de los extraterrestres para que pueda ajustar las dos velocidades de forma independiente.

Para implementar la configuración Fleet_direction, podríamos usar un valor de texto, como como 'izquierda' o 'derecha', pero terminaríamos con sentencias if-elif probando la dirección de la flota. En cambio, debido a que solo tenemos dos direcciones con las que lidiar, usemos los valores 1 y ÿ1, y cambiemos entre ellos cada vez que la flota cambie de dirección. (Usar números también tiene sentido porque moverse a la derecha implica sumar el valor de la coordenada x de cada extraterrestre, y moverse a la izquierda implica restar del valor de la coordenada x de cada extraterrestre).

Comprobando si un extraterrestre ha llegado al borde

Necesitamos un método para verificar si un alienígena está en cualquiera de los bordes, y debemos modificar update() para permitir que cada alienígena se mueva en la dirección adecuada. Este código es parte de la clase Alien :

alien.py

```
def comprobar_bordes(uno mismo):
    """Devolver verdadero si el extraterrestre está en el borde de la pantalla"."""
    screen_rect = self.screen.get_rect()
```

```
en          si self.rect.right >= screen_rect.right o self.rect.left <= 0:
                volver verdadero

        def actualizar (auto):
            """Mueve al alienígena a la derecha o a la izquierda"."""
en          self.x += (self.settings.alien_speed *
                            self.settings.fleet_direction)
            self.rect.x = self.x
```

Podemos llamar al nuevo método check_edges() en cualquier alien para ver si está en el borde izquierdo o derecho. El alien está en el borde derecho si el atributo derecho de su rect es mayor o igual que el atributo derecho del rect de la pantalla . Está en el borde izquierdo si su valor izquierdo es menor o igual a 0 u.

Modificamos el método update() para permitir el movimiento hacia la izquierda o hacia la derecha multiplicando la velocidad del alienígena por el valor de dirección_flota v. Si flota _direction es 1, el valor de alien_speed se agregará a la posición actual del alienígena, moviéndolo hacia la derecha; si la dirección_flota es ÿ1, el valor se restará de la posición del alienígena, moviéndolo hacia la izquierda.

Dejar caer la flota y cambiar de dirección

Cuando un alienígena llega al borde, toda la flota debe descender y cambiar de dirección. Por lo tanto, necesitamos agregar algo de código a AlienInvasion porque ahí es donde comprobaremos si hay extraterrestres en el borde izquierdo o derecho. Haremos que esto suceda escribiendo los métodos _check_fleet_edges() y _change_fleet_direction(), y luego modificando _update_aliens(). Pondré estos nuevos métodos después de _create_alien(), pero nuevamente, la ubicación de estos métodos en la clase no es crítica.

alien_invasion.py

```
        def _check_fleet_edges(auto):
            """Responda adecuadamente si algún extraterrestre ha llegado a un borde"."""
en          para alien en self.aliens.sprites():
                si alien.check_edges():
en                  self._cambiar_dirección_flota()
                    romper

        def _change_fleet_direction(self):
            """Deja caer toda la flota y cambia la dirección de la flota"."""
            para alien en self.aliens.sprites():
en              alien.rect.y += self.settings.fleet_drop_speed
            self.settings.fleet_direction *= -1
```

En _check_fleet_edges(), recorremos la flota y llamamos a check_edges() en cada extranjero u. Si check_edges() devuelve True, sabemos que un extraterrestre está en un borde y toda la flota necesita cambiar de dirección; entonces llamamos _change_fleet _direction() y salimos del bucle v. En _change_fleet_direction(), recorremos todos los alienígenas y soltamos cada uno usando la configuración Fleet_drop _velocidad w; luego cambiamos el valor de Fleet_direction multiplicando su valor actual por ÿ1. La línea que cambia la dirección de la flota no forma parte del bucle for . Queremos cambiar la posición vertical de cada alienígena, pero solo queremos cambiar la dirección de la flota una vez.

Aquí están los cambios a _update_aliens():

alien_invasion.py

```
def _update_aliens(auto):
    """
    Compruebe si la flota está en un borde,
    """luego actualice las posiciones de todos los alienígenas en la flota.

    self._check_fleet_edges()
    self.aliens.update()
```

Hemos modificado el método llamando a _check_fleet_edges() antes actualizando la posición de cada alienígena.

Cuando ejecutas el juego ahora, la flota debe moverse hacia adelante y hacia atrás entre los bordes de la pantalla y caer cada vez que toca un borde.

Ahora podemos comenzar a derribar a los alienígenas y observar a los alienígenas que golpean la nave o alcanzan la parte inferior de la pantalla.

Inténtalo tú mismo

13-3. Gotas de lluvia: encuentre una imagen de una gota de lluvia y cree una cuadrícula de gotas de lluvia. Haz que las gotas de lluvia caigan hacia la parte inferior de la pantalla hasta que desaparezcan.

13-4. Lluvia constante: modifique su código en el ejercicio 13-3 para que cuando una fila de gotas de lluvia desaparezca de la parte inferior de la pantalla, aparezca una nueva fila en la parte superior de la pantalla y comience a caer.

disparando a los extraterrestres

Hemos construido nuestra nave y una flota de alienígenas, pero cuando las balas alcanzan a los alienígenas, simplemente pasan porque no estamos buscando colisiones. En la programación de juegos, las *colisiones* ocurren cuando los elementos del juego se superponen. Para hacer que las balas derriben a los alienígenas, usaremos el método sprite.groupcollide() para buscar colisiones entre miembros de dos grupos.

Detección de colisiones de balas

Queremos saber de inmediato cuándo una bala golpea a un alienígena para que podamos hacer que un alienígena desaparezca tan pronto como sea alcanzado. Para hacer esto, buscaremos colisiones inmediatamente después de actualizar la posición de todas las balas.

La función sprite.groupcollide() compara los rectos de cada elemento en un grupo con los rectos de cada elemento en otro grupo. En este caso, compara el rect de cada bala con el rect de cada alienígena y devuelve un diccionario que contiene las balas y los alienígenas que han chocado. Cada tecla en el

diccionario será una bala, y el valor correspondiente será el alienígena que fue golpeado. (También usaremos este diccionario cuando implementemos un sistema de puntuación en el Capítulo 14).

Agregue el siguiente código al final de _update_bullets() para comprobar si hay colli siones entre balas y extraterrestres:

alien_invasion.py

```
def _update_bullets(auto):
    """Actualizar la posición de las viñetas y deshacerse de las viñetas viejas."""
    --recorte--

    # Verifique si hay balas que hayan golpeado a los extraterrestres.
    # Si es así, deshazte de la bala y del alienígena.
    colisiones = pygame.sprite.groupcollide(
            self.bullets, self.aliens, True, True)
```

El nuevo código que agregamos compara las posiciones de todas las viñetas en self.bullets y todos los alienígenas en self.aliens, e identifica cualquiera que se superponga. Cada vez que los rectos de una viñeta y un alienígena se superponen, groupcollide() agrega un par de valores clave al diccionario que devuelve. Los dos argumentos True le dicen a Pygame que elimine las balas y los alienígenas que han chocado. (Para hacer una bala de alta potencia que pueda viajar a la parte superior de la pantalla, destruyendo a todos los extraterrestres en su camino, puede configurar el primer argumento booleano en Falso y mantener el segundo argumento booleano en Verdadero. El impacto de los alienígenas desaparecería, pero todas las viñetas permanecerían activas hasta que desaparecieran de la parte superior de la pantalla).

Cuando ejecutas *Alien Invasion* ahora, los alienígenas que golpees deberían desaparecer. La figura 13-5 muestra una flota que ha sido derribada parcialmente.

Figura 13-5: ¡Podemos disparar a los extraterrestres!

Fabricación de balas más grandes para pruebas

Puedes probar muchas características del juego simplemente ejecutándolo. Pero algunas características son tediosas de probar en la versión normal del juego. Por ejemplo, es mucho trabajo derribar a todos los extraterrestres en la pantalla varias veces para probar si su código responde correctamente a una flota vacía.

Para probar características particulares, puede cambiar ciertas configuraciones del juego para enfocarse en un área en particular. Por ejemplo, puede reducir la pantalla para que haya menos alienígenas para derribar o aumentar la velocidad de la bala y obtener muchas balas a la vez.

Mi cambio favorito para probar *Alien Invasion* es usar balas realmente anchas que permanecen activas incluso después de haber golpeado a un alienígena (ver Figura 13-6). ¡ Intenta establecer bullet_width en 300, o incluso en 3000, para ver qué tan rápido puedes derribar la flota!

Figura 13-6: Las balas extra poderosas hacen que algunos aspectos del juego sean más fáciles de probar.

Cambios como estos te ayudarán a probar el juego de manera más eficiente y posiblemente generen ideas para otorgar poderes adicionales a los jugadores. Solo recuerde restaurar la configuración a la normalidad cuando termine de probar una función.

Repoblación de la Flota

Una característica clave de *Alien Invasion* es que los alienígenas son implacables: cada vez que se destruye la flota, debería aparecer una nueva flota.

Para hacer que aparezca una nueva flota de alienígenas después de que una flota haya sido destruida, primero verificamos si el grupo de alienígenas está vacío. Si es así, hacemos una llamada a _create_fleet(). Realizaremos esta verificación al final de _update_bullets(), porque ahí es donde se destruyen los extraterrestres individuales.

alien_invasion.py

```
    def _update_bullets(auto):
        --recorte--
en      si no self.aliens:
            # Destruye las balas existentes y crea una nueva flota.
en          self.bullets.empty()
            self._create_fleet()
```

En u, verificamos si el grupo de alienígenas está vacío. Un grupo vacío se evalúa como Falso, por lo que esta es una forma sencilla de verificar si el grupo está vacío. Si es así, nos deshacemos de las viñetas existentes usando el método empty() , que elimina todos los sprites restantes de un grupo v. También llamamos a _create _fleet(), que vuelve a llenar la pantalla con extraterrestres.

Ahora aparece una nueva flota tan pronto como destruyas la flota actual.

Acelerando las balas

Si ha intentado disparar a los alienígenas en el estado actual del juego, es posible que las balas no viajen a la mejor velocidad para el juego. Pueden ser un poco lentos en su sistema o demasiado rápidos. En este punto, puede modificar la configuración para que el juego sea interesante y agradable en su sistema.

Modificamos la velocidad de las balas ajustando el valor de bullet_speed en *configuración.py.* En mi sistema, ajustaré el valor de bullet_speed a 1.5, para que las balas viajen un poco más rápido:

configuración.py

```
        # Configuración de viñetas
        self.bullet_speed = 1.5
        self.bullet_width = 3
        --recorte--
```

El mejor valor para esta configuración depende de la velocidad de su sistema, así que encuentre un valor que funciona para usted. También puede ajustar otras configuraciones.

Refactorización _update_bullets()

Vamos a refactorizar _update_bullets() para que no haga tantas tareas diferentes. Moveremos el código para lidiar con colisiones de balas y extraterrestres a un método separado:

alien_invasion.py

```
    def _update_bullets(auto):
        --recorte--
        # Deshazte de las balas que han desaparecido.
        para viñeta en self.bullets.copy():
            si viñeta.rect.bottom <= 0:
                self.bullets.remove(viñeta)

        self._check_bullet_alien_collisions()

    def _check_bullet_alien_collisions(uno mismo):
        """Responder a colisiones de balas-alienígenas"."""
        # Elimina las balas y los extraterrestres que hayan chocado.
```

```
        colisiones = pygame.sprite.groupcollide(
                self.bullets, self.aliens, True, True)

        si no self.aliens:
            # Destruye las balas existentes y crea una nueva flota.
            self.bullets.empty()
            self._create_fleet()
```

Hemos creado un nuevo método, _check_bullet_alien_collisions(), para buscar colisiones entre balas y alienígenas, y para responder adecuadamente si toda la flota ha sido destruida. Hacerlo evita que _update_bullets() crezca demasiado y simplifica el desarrollo posterior.

Inténtalo tú mismo

13-5. Sideways Shooter Parte 2: Hemos recorrido un largo camino desde el ejercicio 12-6, Sideways Shooter. Para este ejercicio, intente desarrollar Sideways Shooter hasta el mismo punto al que llevamos Alien Invasion. Añade una flota de alienígenas y haz que se muevan lateralmente hacia la nave. O escriba un código que coloque alienígenas en posiciones aleatorias a lo largo del lado derecho de la pantalla y luego los envíe hacia la nave. Además, escriba un código que haga desaparecer a los alienígenas cuando sean golpeados.

Terminar el juego

¿Cuál es la diversión y el desafío en un juego si no puedes perder? Si el jugador no derriba la flota lo suficientemente rápido, haremos que los alienígenas destruyan la nave cuando hagan contacto. Al mismo tiempo, limitaremos la cantidad de barcos que un jugador puede usar y destruiremos el barco cuando un alienígena llegue al final de la pantalla. El juego terminará cuando el jugador haya usado todas sus naves.

Detección de colisiones de naves y extraterrestres

Comenzaremos comprobando las colisiones entre los extraterrestres y la nave para que podamos responder adecuadamente cuando un extraterrestre la golpee. Comprobaremos si hay colisiones entre alienígenas y naves inmediatamente después de actualizar la posición de cada alienígena en AlienInvasion:

alien_invasion.py

```
    def _update_aliens(auto):
        --recorte--
        self.aliens.update()

        # Busque colisiones de naves extraterrestres.
en      if pygame.sprite.spritecollideany(self.ship, self.aliens):
en          print("¡¡Golpe de barco!!!")
```

La función spritecollideany() toma dos argumentos: un sprite y un grupo. La función busca cualquier miembro del grupo que haya colisionado con el sprite y deja de recorrer el grupo en cuanto encuentra un miembro que ha colisionado con el sprite. Aquí, recorre el grupo de extraterrestres y devuelve el primer extraterrestre que encuentra que ha chocado con la nave.

Si no ocurren colisiones, spritecollideany() devuelve Ninguno y el bloque if en u no ejecutará. Si encuentra un alienígena que ha colisionado con la nave, devuelve ese alienígena y ejecuta el bloque if : imprime *Ship hit!!!* v. Cuando un alienígena golpee la nave, tendremos que realizar una serie de tareas: tendremos que eliminar todos los alienígenas y balas restantes, volver a centrar la nave y crear una nueva flota.

Antes de escribir el código para hacer todo esto, debemos saber que nuestro enfoque para detectar colisiones de naves y extraterrestres funciona correctamente. Escribir una llamada print() es una forma sencilla de garantizar que estamos detectando estas colisiones correctamente.

Ahora, cuando ejecutas *Alien Invasion, aparece* el mensaje *Ship hit!!!* debería aparecer en la terminal cada vez que un extraterrestre se encuentre con la nave. Cuando pruebes esta función, establece alien_drop_speed en un valor más alto, como 50 o 100, para que los alienígenas lleguen a tu nave más rápido.

Respuesta a colisiones de naves y extraterrestres

Ahora tenemos que averiguar qué sucederá exactamente cuando un extraterrestre colisione con la nave. En lugar de destruir la instancia de la nave y crear una nueva, contaremos cuántas veces la nave ha sido golpeada mediante el seguimiento de las estadísticas del juego. Las estadísticas de seguimiento también serán útiles para la puntuación.

Escribamos una nueva clase, GameStats, para realizar un seguimiento de las estadísticas del juego y guardarla como *game_stats.py:*

estadísticas_del_juego.py

```
clase GameStats:
    """Estadísticas de seguimiento de Alien Invasion."""

    def __init__(self, ai_game):
        """Inicializar estadísticas."""
        self.settings = ai_game.settings
        self.reset_stats()

    def reset_stats(auto):
        """Inicializar estadísticas que pueden cambiar durante el juego."""
        self.ships_left = self.settings.ship_limit
```

Crearemos una instancia de GameStats para todo el tiempo que se esté ejecutando *Alien Invasion .* Pero necesitaremos restablecer algunas estadísticas cada vez que el jugador comience un nuevo juego. Para hacer esto, inicializaremos la mayoría de las estadísticas en el reinicio _stats() en lugar de directamente en __init__(). Llamaremos a este método desde __init__() para que las estadísticas se establezcan correctamente cuando se crea por primera vez la instancia de GameStats u. Pero también podremos llamar a reset_stats() cada vez que el jugador comience un nuevo juego.

En este momento solo tenemos una estadística, ships_left, cuyo valor será cambiar a lo largo del juego. El número de barcos con los que comienza el jugador debe almacenarse en *settings.py* como ship_limit:

configuración.py

```
# Configuración de envío
self.ship_speed = 1.5
self.ship_limit = 3
```

También necesitamos hacer algunos cambios en *alien_invasion.py* para crear una instancia de GameStats. Primero, actualizaremos las declaraciones de importación en la parte superior del archivo:

alien_invasion.py

```
sistema de importación
desde el tiempo de importación del sueño

importar pygame

desde ajustes importar Ajustes
desde game_stats importar GameStats
desde barco importación Barco
--recorte--
```

Importamos la función sleep() desde el módulo de tiempo en la biblioteca estándar de Python para que podamos pausar el juego por un momento cuando se golpea el barco. También importamos GameStats.

Crearemos una instancia de GameStats en __init__():

alien_invasion.py

```
def __init__(uno mismo):
    --recorte--
    self.pantalla = pygame.display.set_mode(
        (self.settings.screen_width, self.settings.screen_height))
    pygame.display.set_caption("Invasión alienígena")

    # Crea una instancia para almacenar estadísticas del juego.
    self.stats = GameStats(self)

    self.barco = Barco(uno mismo)
    --recorte--
```

Creamos la instancia después de crear la ventana del juego pero antes de definir otros elementos del juego, como la nave.

Cuando un extraterrestre golpee la nave, restaremos uno de la cantidad de naves que quedan, destruiremos todas las balas y alienígenas existentes, crearemos una nueva flota y reubicaremos la nave en el centro de la pantalla. También pausaremos el juego por un momento para que el jugador pueda notar la colisión y reagruparse antes de que aparezca una nueva flota.

Pongamos la mayor parte de este código en un nuevo método llamado _ship_hit(). llamaremos este método de _update_aliens() cuando un extraterrestre golpea la nave:

alien_invasion.py

```
def _ship_hit(auto):
    """Responder a la nave siendo golpeada por un extraterrestre"."""
```

```
        # Decrementar barcos_izquierda.
en      self.stats.ships_left -= 1

        # Deshazte de los alienígenas y las balas restantes.
en      self.aliens.empty()
        self.bullets.empty()

        # Crea una nueva flota y centra el barco.
en      self._create_fleet()
        self.ship.center_ship()

        # Pausa.
X       dormir (0.5)
```

El nuevo método _ship_hit() coordina la respuesta cuando un extraterrestre golpea una nave. Dentro de _ship_hit(), el número de barcos que quedan se reduce en 1 en u, después de lo cual vaciamos los grupos alienígenas y balas v.

A continuación, creamos una nueva flota y centramos el barco w. (Agregaremos el método center_ship() a Enviar en un momento). Luego agregamos una pausa después de que se hayan realizado las actualizaciones en todos los elementos del juego, pero antes de que se hayan dibujado los cambios en la pantalla, para que el jugador pueda ver eso. su nave ha sido golpeada x. La llamada sleep() detiene la ejecución del programa durante medio segundo, el tiempo suficiente para que el jugador vea que el alienígena ha golpeado la nave. Cuando finaliza la función sleep() , la ejecución del código pasa a _update_screen()
método, que dibuja la nueva flota en la pantalla.

En _update_aliens(), reemplazamos la llamada print() con una llamada a _ship_hit() cuando un extraterrestre golpea la nave:

alien_invasion.py

```
def _update_aliens(auto):
    --recorte--
    if pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()
```

Aquí está el nuevo método center_ship(); añádelo al final de *ship.py:*

nave.py

```
def center_ship(auto):
    """Centrar el barco en la pantalla."""
    self.rect.midbottom = self.screen_rect.midbottom
    self.x = float(self.rect.x)
```

Centramos la nave de la misma manera que lo hicimos en __init__(). Después de centrar lo, restablecemos el atributo self.x , que nos permite rastrear la posición exacta del barco.

Tenga en cuenta que nunca fabricamos más de un barco; creamos solo una instancia de nave para todo el juego y la volvemos a actualizar cada vez que la nave ha sido golpeada. La estadística ships_left nos dirá cuando el jugador se haya quedado sin naves.

Ejecuta el juego, dispara a algunos alienígenas y deja que un alienígena golpee la nave. El juego debería hacer una pausa y debería aparecer una nueva flota con el barco centrado en la parte inferior de la pantalla nuevamente.

Alienígenas que llegan al fondo de la pantalla

Si un extraterrestre llega a la parte inferior de la pantalla, haremos que el juego responda de la misma manera que cuando un extraterrestre golpea la nave. Para verificar cuándo sucede esto, agregue un nuevo método en *alien_invasion.py:*

alien_invasion.py

```
def _check_aliens_bottom(uno mismo):
    """Comprueba si algún extraterrestre ha llegado al final de la pantalla."""
    screen_rect = self.screen.get_rect()
    para alien en self.aliens.sprites():
en      si alien.rect.bottom >= screen_rect.bottom:
            # Trate esto de la misma manera que si el barco fuera golpeado.
            self._ship_hit()
            romper
```

El método _check_aliens_bottom() comprueba si algún extraterrestre ha llegado al final de la pantalla. Un extraterrestre alcanza el fondo cuando su valor rect.bottom es mayor o igual al atributo u de rect.bottom de la pantalla . Si un extraterrestre llega al fondo, llamamos _ship_hit(). Si un alienígena toca el fondo, no hay necesidad de comprobar el resto, por lo que salimos del bucle después de llamar a _ship_hit().

Llamaremos a este método desde _update_aliens():

alien_invasion.py

```
def _update_aliens(auto):
    --recorte--
    # Busque colisiones de naves extraterrestres.
    if pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()

    # Busque extraterrestres golpeando la parte inferior de la pantalla.
    self._check_aliens_bottom()
```

Llamamos a _check_aliens_bottom() después de actualizar las posiciones de todos los alienígenas y después de buscar colisiones entre alienígenas y naves v. Ahora aparecerá una nueva flota cada vez que la nave sea golpeada por un alienígena o que un alienígena alcance la parte inferior de la pantalla.

¡Juego terminado!

Alien Invasion se siente más completo ahora, pero el juego nunca termina. El valor de ships_left se vuelve cada vez más negativo. Agreguemos una bandera game_active como un atributo a GameStats para finalizar el juego cuando el jugador se quede sin barcos. Estableceremos esta bandera al final del método __init__() en GameStats:

estadísticas_del_juego.py

```
def __init__(self, ai_game):
    --snip-- #
    Inicia Alien Invasion en un estado activo.
    self.game_active = Verdadero
```

Ahora agregamos código a _ship_hit() que establece game_active en False cuando el el jugador ha usado todas sus naves:

alien_invasion.py

```
def _ship_hit(auto):
    """Responde a la nave siendo golpeada por un extraterrestre"."""
    si self.stats.ships_left > 0:
        # Decrementar barcos_izquierda.
        self.stats.ships_left -= 1
       --recorte--
        # Pausa.
        dormir (0.5)
    demás:
        self.stats.game_active=Falso
```

La mayor parte de _ship_hit() no ha cambiado. Hemos movido todo el código existente a un bloque if , que prueba para asegurarse de que al jugador le quede al menos una nave. Si es así, creamos una nueva flota, hacemos una pausa y seguimos adelante. Si al jugador no le quedan barcos, establecemos game_active en False.

Identificar cuándo deben ejecutarse partes del juego

Necesitamos identificar las partes del juego que siempre deben ejecutarse y las partes que deben ejecutarse solo cuando el juego está activo:

alien_invasion.py

```
def run_game(self):
    """Iniciar el bucle principal del juego."""
    mientras que es cierto:
        self._check_events()

        si self.stats.game_active:
            self.ship.update()
            self._update_bullets()
            self._update_aliens()

        self._update_screen()
```

En el ciclo principal, siempre debemos llamar a _check_events(), incluso si el juego está inactivo. Por ejemplo, todavía necesitamos saber si el usuario presiona Q para salir del juego o si hace clic en el botón para cerrar la ventana. También continuamos actualizando la pantalla para que podamos hacer cambios en la pantalla mientras esperamos ver si el jugador elige comenzar un nuevo juego. El resto de las llamadas a funciones solo deben ocurrir cuando el juego está activo, porque cuando el juego está inactivo, no necesitamos actualizar las posiciones de los elementos del juego.

Ahora, cuando juegues a *Alien Invasion,* el juego debería congelarse cuando hayas usado todas tus naves.

Inténtalo tú mismo

13-6. Game Over: en Sideways Shooter, realiza un seguimiento de la cantidad de veces que la nave es golpeada y la cantidad de veces que la nave golpea a un alienígena. Decide una condición apropiada para terminar el juego y detén el juego cuando ocurra esta situación.

Resumen

En este capítulo, aprendiste a agregar una gran cantidad de elementos idénticos a un juego creando una flota de alienígenas. Usaste bucles anidados para crear una cuadrícula de elementos e hiciste que un gran conjunto de elementos del juego se movieran llamando al método update() de cada elemento. Aprendió a controlar la dirección de los objetos en la pantalla ya responder a situaciones específicas, como cuando la flota llega al borde de la pantalla. Detectaste y respondiste a las colisiones cuando las balas golpearon a los extraterrestres y los extraterrestres golpearon la nave. También aprendió cómo realizar un seguimiento de las estadísticas en un juego y usar una bandera game_active para determinar cuándo termina el juego.

En el siguiente y último capítulo de este proyecto, agregaremos un botón Reproducir para que el jugador puede elegir cuándo comenzar su primer juego y si jugar de nuevo cuando finaliza el juego. Aceleraremos el juego cada vez que el jugador derribe a toda la flota y agregaremos un sistema de puntuación. ¡El resultado final será un juego totalmente jugable!

14

Puntuación

En este capítulo, terminaremos el juego *Alien Invasion* . Agregaremos un botón Reproducir para iniciar un juego a pedido o para reiniciar un juego una vez que finaliza. También cambiaremos el jueg por lo que se acelera cuando el jugador sube de nivel e implementa un sistema de puntuación. Al final del capítulo, sabrá lo suficiente para comenzar a escribir juegos que aumentan en dificultad a medida que el jugador avanza y muestra puntajes.

Agregar el botón de reproducción

En esta sección, agregaremos un botón Reproducir que aparece antes de que comience un juego y reaparece cuando finaliza para que el jugador pueda volver a jugar.

En este momento, el juego comienza tan pronto como ejecutas *alien_invasion.py.* Comencemos el juego en un estado inactivo y luego solicitemos al jugador que haga clic en el botón Reproducir para comenzar. Para ello, modifica el método __init__() de GameStats:

estadísticas_del_juego.py

```
def __init__(self, ai_game):
    """Inicializar estadísticas."""
    self.settings = ai_game.settings
    self.reset_stats()

    # Comienza el juego en un estado inactivo.
    self.game_active = Falso
```

Ahora el juego debería comenzar en un estado inactivo sin forma de que el jugador lo inicie hasta que hagamos un botón de Reproducir.

Crear una clase de botón

Debido a que Pygame no tiene un método integrado para crear botones, escribiremos una clase Button para crear un rectángulo relleno con una etiqueta. Puedes usar este código para hacer cualquier botón en un juego. Aquí está la primera parte de la clase Botón ; guárdelo como *button.py:*

botón.py

```
importar pygame.fuente

Botón de clase:

u def __init__(self, ai_game, msj):
        """Inicializar atributos de botón."""
        self.pantalla = ai_game.pantalla
        self.screen_rect = self.screen.get_rect()

        # Establecer las dimensiones y propiedades del botón.
en      anchura propia, altura propia = 200, 50
        self.button_color = (0, 255, 0)
        self.text_color = (255, 255, 255)
en      self.font = pygame.font.SysFont(Ninguno, 48)

        # Construye el objeto rect del botón y céntralo.
X       self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center

        # El mensaje del botón debe prepararse solo una vez.
y       self._prep_msg(mensaje)
```

Primero, importamos el módulo pygame.font , que permite que Pygame represente texto en la pantalla. El método __init__() toma los parámetros self, el ai_game

object y msg, que contiene el texto del botón u. Establecemos las dimensiones del botón en v, y luego configuramos button_color para colorear el objeto rect del botón de verde brillante y configuramos text_color para representar el texto en blanco.

En w, preparamos un atributo de fuente para representar texto. El argumento Ninguno le dice a Pygame que use la fuente predeterminada y 48 especifica el tamaño del texto. Para centrar el botón en la pantalla, creamos un rect para el botón x y establecemos su atributo central para que coincida con el de la pantalla.

Pygame trabaja con texto representando la cadena que desea mostrar como una imagen. En y, llamamos a _prep_msg() para manejar esta representación.

Aquí está el código para _prep_msg():

botón.py

```
    def _prep_msg(auto, mensaje):
        """Convierta el mensaje en una imagen renderizada y centre el texto en el botón."""
en      self.msg_image = self.font.render(msg, True, self.text_color,
                self.button_color)
en      self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center
```

El método _prep_msg() necesita un autoparámetro y el texto que se representará como una imagen (msg). La llamada a font.render() convierte el texto almacenado en msg en una imagen, que luego almacenamos en self.msg_image u. La fuente.render()
El método también toma un valor booleano para activar o desactivar el antialiasing (el antialiasing suaviza los bordes del texto). Los argumentos restantes son el color de fuente y el color de fondo especificados. Configuramos antialiasing en True
y establezca el fondo del texto en el mismo color que el botón. (Si no incluye un color de fondo, Pygame intentará representar la fuente con un fondo transparente).

En v, centramos la imagen de texto en el botón creando un rect desde la imagen y configurando su atributo central para que coincida con el del botón.

Finalmente, creamos un método draw_button() al que podemos llamar para mostrar el botón en pantalla:

botón.py

```
    def dibujar_boton(auto):
        # Dibujar un botón en blanco y luego dibujar un mensaje.
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

Llamamos a screen.fill() para dibujar la parte rectangular del botón.
Luego llamamos a screen.blit() para dibujar la imagen de texto en la pantalla, pasándole una imagen y el objeto rect asociado con la imagen. Esto completa la clase Button .

Dibujar el botón en la pantalla

Usaremos la clase Button para crear un botón Reproducir en AlienInvasion. Primero, actualizaremos las declaraciones de importación :

alien_invasion.py

```
--recorte--
desde game_stats importar GameStats
desde el botón botón de importación
```

Debido a que solo necesitamos un botón Reproducir, crearemos el botón en el método __init__() de AlienInvasion. Podemos colocar este código al final de __init__():

alien_invasion.py

```
def __init__(uno mismo):
    --recorte--
    self._create_fleet()

    # Haz el botón Reproducir.
    self.play_button = Botón(self, "Reproducir")
```

Este código crea una instancia de Button con la etiqueta *Reproducir,* pero no dibuja el botón en la pantalla. Llamaremos al método draw_button() del botón en _update_screen ():

alien_invasion.py

```
def _update_screen(auto):
    --recorte--
    self.aliens.draw(self.screen)

    # Dibuja el botón de reproducción si el juego está inactivo.
    si no self.stats.game_active:
        self.play_button.draw_button()

    pygame.display.flip()
```

Para que el botón Reproducir sea visible sobre todos los demás elementos de la pantalla, lo dibujamos después de que se hayan dibujado todos los demás elementos, pero antes de pasar a una nueva pantalla. Lo incluimos en un bloque if , por lo que el botón solo aparece cuando el juego está inactivo.

Ahora, cuando ejecute *Alien Invasion,* debería ver un botón Reproducir en el centro de la pantalla, como se muestra en la Figura 14-1.

Figura 14-1: Aparece un botón Jugar cuando el juego está inactivo.

Comenzando el juego

Para comenzar un nuevo juego cuando el jugador hace clic en Jugar, agregue el siguiente elif bloque hasta el final de _check_events() para monitorear los eventos del mouse sobre el botón:

alien_invasion.py

```
    def _check_events(auto):
        """Responder a las pulsaciones de teclas y eventos del ratón."""
        para evento en pygame.event.get():
            if event.type == pygame.QUIT:
                --recorte--
            elif event.type == pygame.MOUSEBUTTONDOWN:
                ratón_pos = pygame.mouse.get_pos()
ultravioleta    self._check_play_button(mouse_pos)
```

Pygame detecta un evento MOUSEBUTTONDOWN cuando el jugador hace clic en cualquier lugar de la pantalla u, pero queremos restringir nuestro juego para que responda a los clics del mouse solo en el botón Reproducir. Para lograr esto, usamos pygame.mouse.get_pos(), que devuelve una tupla que contiene las coordenadas x e y del cursor del mouse cuando se hace clic en el botón del mouse v. Enviamos estos valores al nuevo método _check_play_button() w.

Aquí está _check_play_button(), que elegí colocar después de _check_events():

alien_invasion.py

```
    def _check_play_button(self, mouse_pos):
        """Comience un nuevo juego cuando el jugador haga clic en Jugar"."""
en      si self.play_button.rect.collidepoint(mouse_pos):
            self.stats.game_active = Verdadero
```

Usamos el método rect collidepoint() para comprobar si el punto del clic del ratón se superpone a la región definida por el rect u del botón Reproducir. Si es así, establecemos game_active en True, ¡y comienza el juego!

En este punto, deberías poder comenzar y jugar un juego completo. Cuando finaliza el juego, el valor de game_active debería convertirse en False y el botón Reproducir debería volver a aparecer.

Restablecer el juego

El código del botón Reproducir que acabamos de escribir funciona la primera vez que el jugador hace clic en Reproducir. Pero no funciona después de que finaliza el primer juego, porque las condiciones que causaron el final del juego no se han restablecido.

Para restablecer el juego cada vez que el jugador hace clic en Jugar, debemos restablecer las estadísticas del juego, eliminar los antiguos alienígenas y las balas, construir una nueva flota y centrar la nave, como se muestra aquí:

alien_invasion.py

```
    def _check_play_button(self, mouse_pos):
        """Comience un nuevo juego cuando el jugador haga clic en Jugar"."""
        si self.play_button.rect.collidepoint(mouse_pos):
            # Restablecer las estadísticas del juego.
en          self.stats.reset_stats()
            self.stats.game_active = Verdadero

            # Deshazte de los alienígenas y las balas restantes.
en          self.aliens.empty()
```

```
            self.bullets.empty()

            # Crea una nueva flota y centra el barco.
en          self._create_fleet()
            self.ship.center_ship()
```

En u, reiniciamos las estadísticas del juego, lo que le da al jugador tres nuevos barcos. Luego establecemos game_active en True para que el juego comience tan pronto como el código de esta función termine de ejecutarse. Vaciamos los alienígenas y las balas . grupos v, y luego crea una nueva flota y centra el barco w.

Ahora el juego se reiniciará correctamente cada vez que hagas clic en Jugar, permitiéndote para jugar tantas veces como quieras!

Desactivar el botón de reproducción

Un problema con nuestro botón Reproducir es que la región del botón en la pantalla seguirá respondiendo a los clics incluso cuando el botón Reproducir no esté visible. Si hace clic en el área del botón Reproducir por accidente después de que comience un juego, ¡el juego se reiniciará!

Para solucionar esto, configure el juego para que comience solo cuando game_active es False:

alien_invasion.py

```
    def _check_play_button(self, mouse_pos):
        """Comience un nuevo juego cuando el jugador haga clic en Jugar"."""
en      button_clicked = self.play_button.rect.collidepoint(mouse_pos)
en      si button_clicked y no self.stats.game_active:
            # Restablecer las estadísticas del juego.
            self.stats.reset_stats()
            --recorte--
```

La bandera button_clicked almacena un valor verdadero o falso u, y el juego se reiniciará solo si se hace clic en Reproducir *y* el juego no está actualmente activo v. Para probar este comportamiento, comience un juego nuevo y haga clic repetidamente donde debería estar el botón Reproducir. Si todo funciona como se esperaba, hacer clic en el área del botón Reproducir no debería tener ningún efecto en el juego.

Ocultar el cursor del mouse

Queremos que el cursor del mouse esté visible para comenzar a jugar, pero una vez que comienza, simplemente se interpone en el camino. Para solucionar esto, lo haremos invisible cuando el juego se active. Podemos hacer esto al final del bloque if en _check_play_button():

alien_invasion.py

```
    def _check_play_button(self, mouse_pos):
        """Comience un nuevo juego cuando el jugador haga clic en Jugar"."""
        button_clicked = self.play_button.rect.collidepoint(mouse_pos)
        si button_clicked y no self.stats.game_active:
            --recorte--
            # Ocultar el cursor del mouse.
            pygame.mouse.set_visible(Falso)
```

Pasar False a set_visible() le dice a Pygame que oculte el cursor cuando el el mouse está sobre la ventana del juego.

Haremos que el cursor vuelva a aparecer una vez que finalice el juego para que el jugador pueda haz clic en Reproducir de nuevo para comenzar un nuevo juego. Aquí está el código para hacer eso:

alien_invasion.py

```
def _ship_hit(auto):
    """Responde a la nave siendo golpeada por un extraterrestre".""
    si self.stats.ships_left > 0:
        --recorte--
    demás:
        self.stats.game_active=Falso
        pygame.mouse.set_visible(Verdadero)
```

Volvemos a hacer visible el cursor tan pronto como el juego se vuelve inactivo, lo que sucede en _ship_hit(). La atención a detalles como este hace que su juego se vea más profesional y permite que el jugador se concentre en jugar en lugar de descubrir la interfaz de usuario.

Inténtalo tú mismo

14-1. Presione P para jugar: debido a que Alien Invasion usa la entrada del teclado para controlar la nave, sería útil comenzar el juego presionando una tecla. Agregue un código que le permita al jugador presionar P para comenzar. Podría ayudar mover algo de código de _check _play_button() a un método _start_game() que se puede llamar desde _check_play _button() y _check_keydown_events().

14-2. Práctica de objetivo: cree un rectángulo en el borde derecho de la pantalla que se mueva hacia arriba y hacia abajo a un ritmo constante. Luego, haga que aparezca un barco en el lado izquierdo de la pantalla que el jugador puede mover hacia arriba y hacia abajo mientras dispara balas al objetivo rectangular en movimiento. Agregue un botón Reproducir que inicie el juego, y cuando el jugador falle el objetivo tres veces, finalice el juego y haga que vuelva a aparecer el botón Reproducir. Deje que el jugador reinicie el juego con este botón Reproducir.

Subir de nivel

En nuestro juego actual, una vez que un jugador derriba toda la flota alienígena, el jugador alcanza un nuevo nivel, pero la dificultad del juego no cambia. Animemos un poco las cosas y hagamos que el juego sea más desafiante aumentando la velocidad del juego cada vez que un jugador limpia la pantalla.

Modificación de la configuración de velocidad

Primero reorganizaremos la clase Configuración para agrupar la configuración del juego en estática y variable. También nos aseguraremos de que las configuraciones que cambien

durante el reinicio del juego cuando comenzamos un nuevo juego. Aquí está el __init__() método para *settings.py:*

configuración.py

```
def __init__(uno mismo):
    """Inicializa la configuración estática del juego."""
    # Ajustes de pantalla
    self.screen_width = 1200
    self.screen_height = 800
    self.bg_color = (230, 230, 230)

    # Configuración de envío
    self.ship_limit = 3

    # Configuración de viñetas
    self.bullet_width = 3
    self.bullet_height = 15
    self.bullet_color = 60, 60, 60
    self.bullets_permitido = 3

    # Configuración alienígena
    self.fleet_drop_speed = 10

    # Qué tan rápido se acelera el juego
en  self.speedup_escala = 1.1

en  self.initialize_dynamic_settings()
```

Seguimos inicializando aquellas configuraciones que se mantienen constantes en el __init__() método. En u, agregamos una configuración speedup_scale para controlar qué tan rápido se acelera el juego: un valor de 2 duplicará la velocidad del juego cada vez que el jugador alcanza un nuevo nivel; un valor de 1 mantendrá la velocidad constante. Un valor como 1.1 debería aumentar la velocidad lo suficiente como para que el juego sea desafiante pero no imposible. Finalmente, llamamos al método initialize_dynamic_settings() para inicializar los valores de los atributos que deben cambiar a lo largo del juego v.

Aquí está el código para initialize_dynamic_settings():

configuración.py

```
def initialize_dynamic_settings(auto):
    """Inicializa la configuración que cambia a lo largo del juego."""
    self.ship_speed = 1.5
    self.bullet_speed = 3.0
    self.alien_speed = 1.0

    # Fleet_direction de 1 representa la derecha; -1 representa a la izquierda.
    self.fleet_direction = 1
```

Este método establece los valores iniciales para las velocidades de la nave, la bala y el alienígena. Aumentaremos estas velocidades a medida que el jugador progrese en el juego y las restableceremos cada vez que el jugador comience un nuevo juego. Incluimos flota _dirección en este método para que los alienígenas siempre se muevan justo al comienzo de un nuevo juego. No necesitamos aumentar el valor de fleet_drop_speed, porque cuando los alienígenas se mueven más rápido por la pantalla, también bajarán más rápido por la pantalla.

Para aumentar la velocidad de la nave, las balas y los alienígenas cada vez que el jugador alcance un nuevo nivel, escribiremos un nuevo método llamado Increase_speed():

configuración.py

```
def aumentar_velocidad(auto):
    """Aumentar la configuración de velocidad."""
    self.ship_speed *= self.speedup_scale
    self.bullet_speed *= self.speedup_scale
    self.alien_speed *= self.speedup_scale
```

Para aumentar la velocidad de estos elementos del juego, multiplicamos cada velocidad configuración por el valor de speedup_scale.

Aumentamos el tempo del juego llamando a added_speed () en _check _bullet_alien_collisions() cuando el último alienígena de una flota ha sido derribado:

alien_invasion.py

```
def _check_bullet_alien_collisions(uno mismo):
    --recorte--
    si no self.aliens:
        # Destruye las balas existentes y crea una nueva flota.
        self.bullets.empty()
        self._create_fleet()
        self.settings.aumentar_velocidad()
```

¡ Cambiar los valores de los ajustes de velocidad ship_speed, alien_speed y bullet_speed es suficiente para acelerar todo el juego!

Restablecimiento de la velocidad

Ahora necesitamos devolver cualquier configuración modificada a sus valores iniciales cada vez que el jugador inicia un nuevo juego; de lo contrario, cada nuevo juego comenzaría con la configuración de mayor velocidad del juego anterior:

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    """Comience un nuevo juego cuando el jugador haga clic en Jugar"."""
    button_clicked = self.play_button.rect.collidepoint(mouse_pos)
    si button_clicked y no self.stats.game_active:
        # Restablecer la configuración del juego.
        self.settings.initialize_dynamic_settings()
        --recorte--
```

Jugar a *Alien Invasion* debería ser más divertido y desafiante ahora. Cada vez que limpias la pantalla, el juego debería acelerarse y volverse un poco más difícil. Si el juego se vuelve demasiado difícil demasiado rápido, disminuya el valor de settings.speedup_scale. O si el juego no es lo suficientemente desafiante, aumente el valor ligeramente. Encuentra un punto ideal aumentando la dificultad en un tiempo razonable. El primer par de pantallas debería ser fácil, las siguientes desafiantes pero factibles, y las siguientes pantallas casi imposiblemente difíciles.

Inténtalo tú mismo

14-3. Práctica de objetivo desafiante: comience con su trabajo del ejercicio 14-2 (página 285). Haga que el objetivo se mueva más rápido a medida que avanza el juego y reinicie el objetivo a la velocidad original cuando el jugador haga clic en Reproducir.

14-4. Niveles de dificultad: haga un conjunto de botones para Alien Invasion que le permita al jugador seleccionar un nivel de dificultad inicial apropiado para el juego. Cada botón debe asignar los valores apropiados para los atributos en Configuración necesarios para crear diferentes niveles de dificultad.

Puntuación

Implementemos un sistema de puntuación para rastrear la puntuación del juego en tiempo real y mostrar la puntuación más alta, el nivel y la cantidad de barcos restantes.

La puntuación es una estadística del juego, por lo que agregaremos un atributo de puntuación a GameStats:

estadísticas_del_juego.py

```
clase GameStats:
    --recorte--
    def reset_stats(auto):
        """Inicializar estadísticas que pueden cambiar durante el juego."""
        self.ships_left = self.ai_settings.ship_limit
        puntuación propia = 0
```

Para restablecer la puntuación cada vez que comienza un nuevo juego, inicializamos la puntuación en reset_stats() en lugar de __init__().

Visualización de la partitura

Para mostrar la puntuación en la pantalla, primero creamos una nueva clase, Scoreboard. Por ahora, esta clase solo mostrará la puntuación actual, pero eventualmente la usaremos para informar la puntuación más alta, el nivel y la cantidad de barcos restantes también.

Aquí está la primera parte de la clase; guárdelo como *scoreboard.py:*

marcador.py

```
importar pygame.fuente

marcador de clase:
    """Una clase para reportar información de puntuación."""

u def __init__(self, ai_game):
        """Inicializar atributos de registro de puntajes."""
        self.pantalla = ai_game.pantalla
        self.screen_rect = self.screen.get_rect()
        self.settings = ai_game.settings
        self.stats = ai_game.stats

        # Configuración de fuente para información de puntuación.
        self.text_color = (30, 30, 30)
volvo   self.font = pygame.font.SysFont(Ninguno, 48)
```

```
x       # Prepare la imagen de la partitura inicial.
        self.prep_score()
```

Debido a que Scoreboard escribe texto en la pantalla, comenzamos importando el módulo pygame.font . Luego, le damos a __init__() el parámetro ai_game para que pueda acceder a los objetos de configuración, pantalla y estadísticas , que necesitará para informar los valores que estamos rastreando. Luego establecemos un color de texto v e instanciamos un objeto de fuente w.

Para convertir el texto que se mostrará en una imagen, llamamos a prep_score() x, que definimos aquí:

marcador.py

```
    def prep_score(self):
        """Convierte la partitura en una imagen renderizada."""
        score_str = str(self.stats.score)
ultravioleta    self.score_image = self.font.render(score_str, True,
                self.text_color, self.settings.bg_color)

        # Muestra la puntuación en la parte superior derecha de la pantalla.
        self.score_rect = self.score_image.get_rect()
        self.score_rect.right = self.screen_rect.right - 20
wxy     self.score_rect.top = 20
```

En prep_score(), convertimos el valor numérico stats.score en una cadena u, y luego pasamos esta cadena a render(), que crea la imagen v. Para mostrar la puntuación claramente en pantalla, pasamos el color de fondo de la pantalla y el texto color para renderizar().

Colocaremos el puntaje en la esquina superior derecha de la pantalla y haremos que se expanda hacia la izquierda a medida que aumenta el puntaje y crece el ancho del número. Para asegurarnos de que la puntuación siempre se alinee con el lado derecho de la pantalla, creamos un rect llamado score_rect w y establecemos su borde derecho a 20 píxeles del borde derecho de la pantalla x. Luego colocamos el borde superior 20 píxeles hacia abajo desde la parte superior de la pantalla y.

Luego creamos un método show_score() para mostrar la imagen de la partitura renderizada:

marcador.py

```
    def show_score(self):
        """Dibujar puntuación en la pantalla."""
        self.screen.blit(self.score_image, self.score_rect)
```

Este método dibuja la imagen de la partitura en la pantalla en la ubicación score_rect especifica.

Hacer un marcador

Para mostrar la puntuación, crearemos una instancia de Scoreboard en AlienInvasion. Primero, actualicemos las declaraciones de importación :

alien_invasion.py

```
--recorte--
desde game_stats importar GameStats
desde el marcador importar el marcador
--recorte--
```

A continuación, creamos una instancia de Scoreboard en __init__():

alien_invasion.py

```
def __init__(self): --snip--

    pygame.display.set_caption("Invasión alienígena")

    # Cree una instancia para almacenar estadísticas del juego, #
    y cree un marcador. self.stats = GameStats(self) self.sb =
    Scoreboard(self) --snip--
```

Luego dibujamos el marcador en pantalla en _update_screen():

alien_invasion.py

```
def _update_screen(self): --snip--
    self.aliens.draw(self.screen)

    # Dibuja la información de la puntuación.
    self.sb.show_score()

    # Dibuja el botón de reproducción si el juego está inactivo. --
    recorte--
```

Llamamos a show_score() justo antes de dibujar el botón Reproducir.

Cuando ejecutes *Alien Invasion* ahora, debería aparecer un 0 en la parte superior derecha de la pantalla. (En este punto, solo queremos asegurarnos de que el puntaje aparezca en el lugar correcto antes de desarrollar más el sistema de puntaje). La figura 14-2 muestra el puntaje tal como aparece antes de que comience el juego.

Figura 14-2: La puntuación aparece en la esquina superior derecha de la pantalla.

¡A continuación, asignaremos valores de puntos a cada alienígena!

Actualización de la puntuación a medida que los alienígenas son derribados

Para escribir una puntuación en vivo en la pantalla, actualizamos el valor de stats.score cada vez que se golpea a un alienígena y luego llamamos a prep_score() para actualizar la imagen de la puntuación. Pero primero, determinemos cuántos puntos obtiene un jugador cada vez que derriba a un alienígena:

configuración.py

```
def initialize_dynamic_settings(auto):
    --recorte--

    # Puntuación
    self.puntos_alienígenas = 50
```

Aumentaremos el valor en puntos de cada alienígena a medida que avanza el juego. Para asegurarnos de que este valor de punto se restablece cada vez que comienza un nuevo juego, establecemos el valor en initialize_dynamic_settings().

Actualicemos la puntuación cada vez que derriben a un alienígena en _check_bullet _colisiones_alienígenas():

alien_invasion.py

```
def _check_bullet_alien_collisions(uno mismo):
    """Responder a colisiones de balas-alienígenas"."""
    # Elimina las balas y los extraterrestres que hayan chocado.
    colisiones = pygame.sprite.groupcollide(
            self.bullets, self.aliens, True, True)

    si colisiones:
        self.stats.score += self.settings.alien_points
        self.sb.prep_score()
    --recorte--
```

Cuando una bala golpea a un extraterrestre, Pygame devuelve un diccionario de colisiones . Verificamos si el diccionario existe, y si existe, el valor del extraterrestre se suma a la puntuación. Luego llamamos a prep_score() para crear una nueva imagen para la puntuación actualizada.

¡Ahora, cuando juegues a *Alien Invasion,* deberías poder acumular puntos!

Restablecimiento de la puntuación

En este momento, solo estamos preparando una nueva partitura *después de* que un alienígena haya sido golpeado, lo que funciona durante la mayor parte del juego. Pero todavía vemos el puntaje anterior cuando comienza un nuevo juego hasta que el primer alienígena es golpeado en el nuevo juego.

Podemos arreglar esto preparando la puntuación al comenzar un nuevo juego:

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    --recorte--
    si button_clicked y no self.stats.game_active:
        --recorte--
        # Restablecer las estadísticas del juego.
        self.stats.reset_stats()
```

```
        self.stats.game_active = Verdadero
        self.sb.prep_score()
        --recorte--
```

Llamamos a prep_score() después de restablecer las estadísticas del juego al iniciar un nuevo juego. Esto prepara el marcador con una puntuación de 0.

Asegurarse de anotar todos los golpes

Tal como está escrito actualmente, nuestro código podría perder la puntuación de algunos alienígenas. Por ejemplo, si dos balas chocan con alienígenas durante el mismo paso por el bucle o si hacemos una bala extra ancha para golpear a varios alienígenas, el jugador solo recibirá puntos por golpear a uno de los alienígenas. Para arreglar esto, refinemos la forma en que se detectan las colisiones de balas y extraterrestres.

En _check_bullet_alien_collisions(), cualquier bala que choca con un extraterrestre se convierte en una clave en el diccionario de colisiones . El valor asociado con cada bala es una lista de alienígenas con los que ha chocado. Recorremos los valores en el diccionario de colisiones para asegurarnos de otorgar puntos por cada impacto alienígena:

alien_invasion.py

```
def _check_bullet_alien_collisions(uno mismo):
    --recorte--
    si colisiones:
en      para extraterrestres en colisiones.valores():
            self.stats.score += self.settings.alien_points * len(extranjeros)
        self.sb.prep_score()
    --recorte--
```

Si se ha definido el diccionario de colisiones , recorremos todos los valores del diccionario. Recuerda que cada valor es una lista de alienígenas alcanzados por una sola bala. Multiplicamos el valor de cada extranjero por el número de extranjeros en cada lista y sumamos esta cantidad al puntaje actual. Para probar esto, cambie el ancho de una bala a 300 píxeles y verifique que recibe puntos por cada alienígena que golpea con sus balas extra anchas; luego devuelva el ancho de viñeta a su valor normal.

Valores de puntos crecientes

Debido a que el juego se vuelve más difícil cada vez que un jugador alcanza un nuevo nivel, los extraterrestres en niveles posteriores deberían valer más puntos. Para implementar esta funcionalidad, agregaremos código para aumentar el valor del punto cuando la velocidad del juego aumenta:

configuración.py

```
Configuración de clase:
    """Una clase para almacenar todos los ajustes de Alien Invasion."""

    def __init__(uno mismo):
        --recorte--
        # Qué tan rápido se acelera el juego
        self.speedup_escala = 1.1
```

```
en          # Qué tan rápido aumentan los valores de los puntos alienígenas
            self.score_scale = 1.5

            self.initialize_dynamic_settings()

        def initialize_dynamic_settings(auto):
            --recorte--

        def aumentar_velocidad(auto):
            """Aumenta la configuración de velocidad y los valores de puntos alienígenas"."""
            self.ship_speed *= self.speedup_scale
            self.bullet_speed *= self.speedup_scale
            self.alien_speed *= self.speedup_scale

en          self.puntos_alienígenas = int(self.puntos_alienígenas * self.escala_puntuación)
```

Definimos una tasa a la que aumentan los puntos, a la que llamamos score_scale u. Un pequeño aumento en la velocidad (1.1) hace que el juego sea más desafiante rápidamente. Pero para ver una diferencia más notable en la puntuación, necesitamos cambiar el valor del punto alienígena en una cantidad mayor (1,5). Ahora, cuando aumentamos la velocidad del juego, también aumentamos el valor de puntos de cada golpe v. Usamos la función int() para aumentar el valor de puntos en números enteros.

Para ver el valor de cada extraterrestre, agregue una llamada print() a la función de aumento_velocidad() método en Configuración:

configuración.py

```
        def aumentar_velocidad(auto):
            --recorte--
            self.puntos_alienígenas = int(self.puntos_alienígenas * self.escala_puntuación)
            imprimir(self.puntos_alienígenas)
```

El nuevo valor de puntos debería aparecer en la terminal cada vez que alcances un nuevo nivel.

Asegúrese de eliminar la llamada *print() después de verificar que el valor del punto está aumentando, o podría afectar el rendimiento de su juego y distraer al jugador.*

Redondeando la puntuación

La mayoría de los juegos de disparos de estilo arcade reportan puntajes como múltiplos de 10, así que sigamos esa pista con nuestros puntajes. Además, formateemos la partitura para incluir separadores de coma en números grandes. Haremos este cambio en el marcador:

marcador.py

```
        def prep_score(self):
            """Convierte la partitura en una imagen renderizada."""
en          rounded_score = round(self.stats.score, -1)
en          score_str = "{:,}". formato (puntuación_redondeada)
            self.score_image = self.font.render(score_str, True,
                    self.text_color, self.settings.bg_color)
            --recorte--
```

La función round() normalmente redondea un número decimal a un número fijo de lugares decimales dado como segundo argumento. Sin embargo, cuando pasa un número negativo como segundo argumento, round() redondeará el valor al 10, 100, 1000 más cercano, y así sucesivamente. El código en u le dice a Python que redondee el valor de stats.score al 10 más cercano y lo almacene en rounded_score.

En v, una directiva de formato de cadena le dice a Python que inserte comas en números al convertir un valor numérico en una cadena: por ejemplo, para generar 1,000,000 en lugar de 1000000. Ahora, cuando ejecute el juego, debería ver una puntuación redondeada y con un formato ordenado, incluso cuando acumule muchos puntos, como se muestra en la Figura 14 -3.

Figura 14-3: Una puntuación redondeada con separadores de coma

Puntuaciones altas

Todos los jugadores quieren superar la puntuación más alta de un juego, así que hagamos un seguimiento e informemos las puntuaciones más altas para darles a los jugadores algo por lo que trabajar. Guardaremos puntuaciones altas en Estadísticas del juego:

estadísticas_del_juego.py

```
def __init__(self, ai_game):
    --recorte--
    # La puntuación alta nunca debe restablecerse.
    self.high_score = 0
```

Debido a que la puntuación más alta nunca debe restablecerse, inicializamos high_score en __init__() en lugar de reset_stats().

A continuación, modificaremos el marcador para mostrar la puntuación más alta. Comencemos con el método __init__() :

marcador.py

```
def __init__(self, ai_game):
    --recorte--
    # Prepare las imágenes de la partitura inicial.
    self.prep_score()
en  self.prep_high_score()
```

La puntuación más alta se mostrará por separado de la puntuación, por lo que necesitamos un nuevo método, prep_high_score(), para preparar la imagen de puntuación más alta u.

Aquí está el método prep_high_score() :

marcador.py

```
def prep_high_score(self):
    """Convierte la puntuación más alta en una imagen renderizada"."""
en  high_score = round(self.stats.high_score, -1)
    puntuación_alta_str = "{:,}".format(puntuación_alta)
en  self.high_score_image = self.font.render(high_score_str, True,
            self.text_color, self.settings.bg_color)

    # Centre la puntuación más alta en la parte superior de la pantalla.
    self.high_score_rect = self.high_score_image.get_rect()
en  self.high_score_rect.centerx = self.screen_rect.centerx
X   self.high_score_rect.top = self.score_rect.top
```

Redondeamos el puntaje más alto al 10 más cercano y lo formateamos con comas u. A continuación, generamos una imagen a partir de la puntuación más alta v, centramos la puntuación más alta en el recto horizontal w y establecemos su atributo superior para que coincida con la parte superior de la imagen de puntuación x.

El método show_score() ahora dibuja la puntuación actual en la parte superior derecha y la puntuación más alta en la parte superior central de la pantalla:

marcador.py

```
def show_score(self):
    """Dibujar puntuación en la pantalla."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
```

Para comprobar las puntuaciones altas, escribiremos un nuevo método, check_high_score(), en el marcador:

marcador.py

```
def check_high_score(self):
    """Verifique si hay una nueva puntuación alta"."""
    if self.stats.score > self.stats.high_score:
        self.stats.high_score = self.stats.score
        self.prep_high_score()
```

El método check_high_score() compara la puntuación actual con la puntuación más alta. Si la puntuación actual es mayor, actualizamos el valor de high_score y llame a prep_high_score() para actualizar la imagen de la puntuación más alta.

Necesitamos llamar a check_high_score() cada vez que un alienígena es golpeado después de la actualización ing la puntuación en _check_bullet_alien_collisions():

alien_invasion.py

```
def _check_bullet_alien_collisions(uno mismo):
    --recorte--
    si colisiones:
        para extraterrestres en colisiones.valores():
            self.stats.score += self.settings.alien_points * len(extranjeros)
        self.sb.prep_score()
        self.sb.check_high_score()
    --recorte--
```

Llamamos a check_high_score() cuando el diccionario de colisiones está presente, y lo hacemos después de actualizar la puntuación de todos los alienígenas que han sido golpeados.

La primera vez que juegues a *Alien Invasion,* tu puntaje será el puntaje más alto, por lo que se mostrará como el puntaje actual y el puntaje más alto. Pero cuando comienza un segundo juego, su puntaje más alto debería aparecer en el medio y su puntaje actual a la derecha, como se muestra en la Figura 14-4.

Figura 14-4: La puntuación más alta se muestra en la parte superior central de la pantalla.

Visualización del nivel

Para mostrar el nivel del jugador en el juego, primero necesitamos un atributo en GameStats que represente el nivel actual. Para restablecer el nivel al comienzo de cada nuevo juego, inicialícelo en reset_stats():

estadísticas_del_juego.py

```
def reset_stats(auto):
    """Inicializar estadísticas que pueden cambiar durante el juego."""
    self.ships_left = self.settings.ship_limit
```

```
auto.puntaje = 0
auto.nivel = 1
```

Para que Scoreboard muestre el nivel actual, llamamos a un nuevo método, prep _level(), de __init__():

marcador.py

```
def __init__(self, ai_game): --
    snip-- self.prep_high_score()
    self.prep_level()
```

Aquí está prep_level():

marcador.py

```
    def prep_level(self):
        """Convierte el nivel en una imagen renderizada."""
        level_str = str(self.stats.level) self.level_image =
en      self.font.render(level_str, True, self.text_color, self.
                configuración.bg_color)

        # Coloque el nivel debajo de la puntuación.
        self.level_rect = self.level_image.get_rect()
        self.level_rect.right = self.score_rect.right
volvo   self.level_rect.top = self.score_rect.bottom + 10
```

El método prep_level() crea una imagen a partir del valor almacenado en stats.level u y establece el atributo derecho de la imagen para que coincida con el atributo derecho de la puntuación v. Luego establece el atributo superior 10 píxeles debajo de la parte inferior de la imagen de la puntuación para dejar espacio entre la puntuación y el nivel w.

También necesitamos actualizar show_score():

marcador.py

```
def show_score(self):
    """Dibuja puntuaciones y nivela la pantalla."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect) self.
    screen.blit(self.level_image, self.level_rect)
```

Esta nueva línea dibuja la imagen del nivel en la pantalla.

Incrementaremos stats.level y actualizaremos la imagen de nivel en _check_bullet _alien_collisions():

alien_invasion.py

```
def _check_bullet_alien_collisions(self): --snip--
    if not self.aliens: # Destruye las balas
    existentes y crea una nueva flota.
        self.bullets.empty() self._create_fleet()
        self.settings.increase_speed()



        # Aumentar el nivel.
        self.stats.level += 1
        self.sb.prep_level()
```

Si se destruye una flota, incrementamos el valor de stats.level y llamamos prep_level() para asegurarse de que el nuevo nivel se muestre correctamente.

Para garantizar que la imagen del nivel se actualice correctamente al comienzo de un nuevo juego, también llamamos a prep_level() cuando el jugador hace clic en el botón Reproducir:

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    --recorte--
    si button_clicked y no self.stats.game_active:
        --recorte--
        self.sb.prep_score()
        self.sb.prep_level()
        --recorte--
```

Llamamos a prep_level() justo después de llamar a prep_score().

Ahora verá cuántos niveles ha completado, como se muestra en la Figura 14-5.

Figura 14-5: El nivel actual aparece justo debajo de la puntuación actual.

En algunos juegos clásicos, las puntuaciones tienen etiquetas, como Puntuación, Puntuación alta y Nivel. Hemos omitido estas etiquetas porque el significado de cada número queda claro una vez que has jugado el juego. Para incluir estas etiquetas, agréguelas a las cadenas de puntuación justo antes de las llamadas a font.render() en Scoreboard.

Visualización del número de barcos

Finalmente, mostremos la cantidad de barcos que le quedan al jugador, pero esta vez, usemos un gráfico. Para hacerlo, dibujaremos barcos en la esquina superior izquierda de

la pantalla para representar cuántos barcos quedan, tal como lo hacen muchos juegos de arcade clásicos.

Primero, necesitamos hacer que Ship herede de Sprite para que podamos crear un grupo de barcos:

nave.py

```
importar pygame
de pygame.sprite importar Sprite

Barco de clase u (Sprite):
    """Una clase para manejar la nave."""

    def __init__(self, ai_game):
        """Inicializar el barco y establecer su posición inicial."""
en      super().__init__()
        --recorte--
```

Aquí importamos Sprite, nos aseguramos de que Ship herede de Sprite u, y llamamos a super() al comienzo de __init__() v.

A continuación, debemos modificar el marcador para crear un grupo de barcos que podamos mostrar. Aquí están las declaraciones de importación para Scoreboard:

marcador.py

```
importar pygame.fuente
del grupo de importación pygame.sprite

desde barco importación Barco
```

Debido a que estamos haciendo un grupo de barcos, importamos Group and Ship clases

Aquí está __init__():

marcador.py

```
    def __init__(self, ai_game):
        """Inicializar atributos de registro de puntajes."""
        self.ai_game = ai_game
        self.pantalla = ai_game.pantalla
        --recorte--
        self.prep_level()
        self.prep_ships()
```

Asignamos la instancia del juego a un atributo, porque lo necesitaremos para crear algunos barcos. Llamamos a prep_ships() después de la llamada a prep_level().

Aquí está prep_ships():

marcador.py

```
    def prep_ships(auto):
        """Mostrar cuántos barcos quedan."""
        self.barcos = Grupo()
ultravioleta para ship_number en el rango (self.stats.ships_left):
            barco = Barco(self.ai_game)
            barco.rect.x = 10 + número_barco * barco.rect.ancho
            nave.rect.y = 10
wxy         self.barcos.añadir(barco)
```

El método prep_ships() crea un grupo vacío, self.ships, para contener las instancias de envío u. Para llenar este grupo, se ejecuta un ciclo una vez por cada barco que el jugador ha dejado v. Dentro del ciclo, creamos un nuevo barco y establecemos el valor de la coordenada x de cada barco para que los barcos aparezcan uno al lado del otro con un margen de 10 píxeles en el lado izquierdo del grupo de naves w. Establecemos el valor de la coordenada y 10 píxeles hacia abajo desde la parte superior de la pantalla para que los barcos aparezcan en la esquina superior izquierda de la pantalla x. Luego agregamos cada barco nuevo al grupo de barcos y.

Ahora necesitamos dibujar los barcos en la pantalla:

marcador.py

```
def show_score(self):
    """Dibuja puntuaciones, niveles y barcos en la pantalla."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)
    self.ships.draw(self.screen)
```

Para mostrar los barcos en la pantalla, llamamos a draw() en el grupo y Pygame dibuja cada barco.

Para mostrarle al jugador con cuántos barcos tiene para comenzar, llamamos a prep_ships() cuando comienza un nuevo juego. Hacemos esto en _check_play_button() en AlienInvasion:

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    --recorte--
    si button_clicked y no self.stats.game_active:
        --recorte--
        self.sb.prep_score()
        self.sb.prep_level()
        self.sb.prep_ships()
        --recorte--
```

También llamamos a prep_ships() cuando se golpea un barco para actualizar la visualización del barco . imágenes cuando el jugador pierde un barco:

alien_invasion.py

```
def _ship_hit(auto):
    """Responde a la nave siendo golpeada por un extraterrestre"."""
    si self.stats.ships_left > 0:
        # Decrementa los barcos a la izquierda y actualiza el marcador.
        self.stats.ships_left -= 1
        self.sb.prep_ships()
        --recorte--
```

Llamamos a prep_ships() después de disminuir el valor de ships_left, por lo que el el número correcto de barcos se muestra cada vez que se destruye un barco.

La Figura 14-6 muestra el sistema de puntuación completo con los barcos restantes que se muestra en la parte superior izquierda de la pantalla.

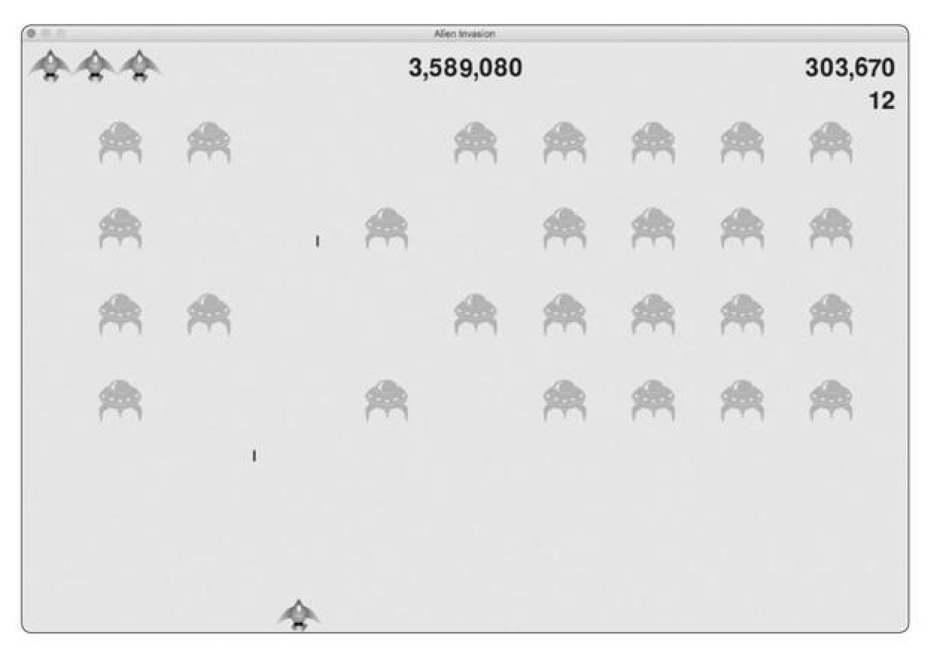


Figura 14-6: El sistema de puntuación completo para Alien Invasion

Inténtalo tú mismo

14-5. Puntuación más alta de todos los tiempos: la puntuación más alta se restablece cada vez que un jugador cierra y reinicia Alien Invasion. Solucione esto escribiendo la puntuación más alta en un archivo antes de llamar a sys.exit() y leyendo la puntuación más alta al inicializar su valor en GameStats.

14-6. Refactorización: busque métodos que realicen más de una tarea y refactorícelos para organizar su código y hacerlo eficiente. Por ejemplo, mueva parte del código en _check_bullet_alien_collisions(), que inicia un nuevo nivel cuando la flota de alienígenas ha sido destruida, a una función llamada start_new _nivel(). Además, mueva las cuatro llamadas de método separadas en el método __init__() en Scoreboard a un método llamado prep_images() para acortar __init__(). El método prep_images() también podría ayudar a simplificar _check_play_button() o iniciar _game() si ya ha refactorizado _check_play_button().

nota Antes de intentar refactorizar el proyecto, consulte el Apéndice D para saber cómo restaurar el proyecto a un estado de funcionamiento si introduce errores durante la refactorización.

(continuado)

14-7. Expansión del juego: piensa en una forma de expandir Alien Invasion. Por ejemplo, puede programar a los alienígenas para que disparen balas a la nave o agregue escudos para que su nave se esconda detrás, que pueden ser destruidos por balas desde cualquier lado. O use algo como el módulo pygame.mixer para agregar efectos de sonido, como explosiones y sonidos de disparos.

14-8. Sideways Shooter, versión final: continúa desarrollando Sideways Shooter, usando todo lo que hemos hecho en este proyecto. Agregue un botón Reproducir, haga que el juego se acelere en los puntos apropiados y desarrolle un sistema de puntuación. Asegúrese de refactorizar su código mientras trabaja y busque oportunidades para personalizar el juego más allá de lo que se muestra en este capítulo.

Resumen

En este capítulo, aprendió cómo implementar un botón Reproducir para comenzar un nuevo juego, detectar eventos del mouse y ocultar el cursor en juegos activos. Puede usar lo que ha aprendido para crear otros botones en sus juegos, como un botón de Ayuda para mostrar instrucciones sobre cómo jugar. También aprendió cómo modificar la velocidad de un juego a medida que avanza, implementar un sistema de puntaje progresivo y mostrar información en formas textuales y no textuales.

Proyecto 2

Visualización de datos



15

Genera nuestros Datos

La visualización de datos implica explorar datos a través de representaciones visuales. Está estrechamente relacionado con *el análisis de datos,* que utiliza código para explorar los patrones y las conexiones en un análisis de datos . colocar. Un conjunto de datos puede estar formado por una pequeña lista de números que caben en una línea de código o pueden ser muchos gigabytes de datos.

Hacer hermosas representaciones de datos es más que imágenes bonitas. Cuando una representación de un conjunto de datos es simple y visualmente atractiva, su significado queda claro para los espectadores. Las personas verán patrones e importancia en sus conjuntos de datos que nunca supieron que existían.

Afortunadamente, no necesita una supercomputadora para visualizar datos complejos. Con la eficiencia de Python, puede explorar rápidamente conjuntos de datos compuestos por millones de puntos de datos individuales en solo una computadora portátil. Además, los puntos de datos no tienen que ser números. Con los conceptos básicos que aprendió en la primera parte de este libro, también puede analizar datos no numéricos.

La gente usa Python para trabajos intensivos en datos en genética, investigación climática, análisis político y económico, y mucho más. Los científicos de datos han escrito

una impresionante variedad de herramientas de visualización y análisis en Python, muchas de las cuales también están disponibles para usted. Una de las herramientas más populares es Matplotlib, una biblioteca de gráficos matemáticos. Usaremos Matplotlib para hacer diagramas simples, como gráficos de líneas y diagramas de dispersión. Luego, crearemos un conjunto de datos más interesante basado en el concepto de recorrido aleatorio: una visualización generada a partir de una serie de decisiones aleatorias.

También usaremos un paquete llamado Plotly, que crea visualizaciones que funcionan bien en dispositivos digitales. Plotly genera visualizaciones que cambian de tamaño automáticamente para adaptarse a una variedad de dispositivos de visualización. Estas visualizaciones también pueden incluir una serie de características interactivas, como enfatizar aspectos particulares del conjunto de datos cuando los usuarios se desplazan sobre diferentes partes de la visualización. Usaremos Plotly para analizar los resultados de tirar los dados.

Instalación de Matplotlib

Para usar Matplotlib para su conjunto inicial de visualizaciones, deberá instalarlo usando pip, un módulo que descarga e instala paquetes de Python. Ingrese el siguiente comando en un indicador de terminal:

```
$ python -m instalación pip --user matplotlib
```

Este comando le dice a Python que ejecute el módulo pip e instale el paquete matplotlib en la instalación de Python del usuario actual. Si usa un comando que no sea python en su sistema para ejecutar programas o iniciar una sesión de terminal, como python3, su comando se verá así:

```
$ python3 -m pip instalar --usuario matplotlib
```

Si este comando no funciona en macOS, intente ejecutar el comando nuevamente sin la marca --user .

Para ver los tipos de visualizaciones que puede realizar con Matplotlib, visite la galería de muestras en *https:// matplotlib.org/ gallery/.* Cuando hace clic en una visualización en la galería, verá el código utilizado para generar el gráfico.

Trazar un gráfico de línea simple

Tracemos un gráfico de líneas simple usando Matplotlib y luego personalícelo para crear una visualización de datos más informativa. Usaremos la secuencia de números cuadrados 1, 4, 9, 16, 25 como datos para el gráfico.

Simplemente proporcione Matplotlib con los números, como se muestra aquí, y Matplotlib debería hacer el resto:

mpl_cuadrados.py

```
importar matplotlib.pyplot como plt

cuadrados = [1, 4, 9, 16, 25]
```

```
ÿ fig, ax = plt.subparcelas()
  ax.plot(cuadrados)

  plt.mostrar()
```

Primero importamos el módulo pyplot usando el alias plt para no tener que escribir pyplot repetidamente. (Verá esta convención a menudo en ejemplos en línea, así que haremos lo mismo aquí.) El módulo pyplot contiene una serie de funciones que generan gráficos y diagramas.

Creamos una lista llamada cuadrados para contener los datos que trazaremos. Entonces nosotros siga otra convención común de Matplotlib llamando a subplots() función ÿ. Esta función puede generar uno o más gráficos en la misma figura. La variable fig representa la figura completa o colección de gráficos que se generan. La variable ax representa una sola parcela en la figura y es la variable que usaremos la mayor parte del tiempo.

Luego usamos el método plot() , que intentará graficar los datos proporcionados de una manera significativa. La función plt.show() abre el visor de Matplotlib y muestra el diagrama, como se muestra en la Figura 15-1. El visor le permite hacer zoom y navegar por el gráfico, y cuando hace clic en el icono del disco, puede guardar las imágenes del gráfico que desee.

Figura 15-1: Uno de los gráficos más simples que puede hacer en Matplotlib

Cambiar el tipo de etiqueta y el grosor de línea

Aunque el gráfico de la figura 15-1 muestra que los números aumentan, el tipo de etiqueta es demasiado pequeño y la línea es un poco delgada para leerla con facilidad. Afortunadamente, Matplotlib le permite ajustar todas las características de una visualización.

Usaremos algunas de las personalizaciones disponibles para mejorar la legibilidad de este gráfico, como se muestra aquí:

mpl_cuadrados.py

```
importar matplotlib.pyplot como plt

cuadrados = [1, 4, 9, 16, 25]

higo, hacha = plt.subplots()
ÿ ax.plot(cuadrados, ancho de línea=3)

# Establecer el título del gráfico y los ejes de etiquetas.
ÿ ax.set_title("Números cuadrados", tamaño de fuente=24)
ÿ ax.set_xlabel("Valor", tamaño de fuente=14)
ax.set_ylabel("Cuadrado de valor", tamaño de fuente=14)

# Establecer el tamaño de las etiquetas de marca.
ÿ ax.tick_params(axis='ambos', labelsize=14)

plt.mostrar()
```

El parámetro linewidth en ÿ controla el grosor de la línea que genera plot() . El método set_title() en ÿ establece un título para el gráfico. Los parámetros de tamaño de fuente, que aparecen repetidamente en todo el código, controlan el tamaño del texto en varios elementos del gráfico.

Los métodos set_xlabel() y set_ylabel() le permiten establecer un título para cada uno de los ejes ÿ, y el método tick_params() le da estilo a las marcas de verificación ÿ. Los argumentos que se muestran aquí afectan las marcas de verificación en los ejes x e y (eje = 'ambos') y establecen el tamaño de fuente de las etiquetas de marca de verificación en 14 (tamaño de etiqueta = 14).

Como puede ver en la Figura 15-2, el gráfico resultante es mucho más fácil de leer. El tipo de etiqueta es más grande y el gráfico de líneas es más grueso. A menudo vale la pena experimentar con estos valores para tener una idea de lo que se verá mejor en el gráfico resultante.

Figura 15-2: El gráfico es mucho más fácil de leer ahora.

Corrección de la trama

Pero ahora que podemos leer mejor el gráfico, vemos que los datos no se trazan correctamente. ¡Observe al final del gráfico que el cuadrado de 4.0 se muestra como 25! Arreglemos eso.

Cuando le da a plot() una secuencia de números, asume que el primer punto de datos corresponde a un valor de coordenada x de 0, pero nuestro primer punto corresponde a un valor de x de 1. Podemos anular el comportamiento predeterminado dando plot () los valores de entrada y salida utilizados para calcular los cuadrados:

mpl_cuadrados.py

```
importar matplotlib.pyplot como plt

valores_de_entrada = [1, 2, 3, 4, 5]
cuadrados = [1, 4, 9, 16, 25]

higo, hacha = plt.subplots()
ax.plot(valores_de_entrada, cuadrados, ancho de línea=3)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--
```

Ahora , plot() graficará los datos correctamente porque proporcionamos los valores de entrada y salida, por lo que no tiene que asumir cómo se generaron los números de salida. El gráfico resultante, que se muestra en la figura 15-3, es correcto.

Figura 15-3: Los datos ahora se trazan correctamente.

Puede especificar numerosos argumentos al usar plot() y usar un número ber de funciones para personalizar sus parcelas. Continuaremos explorando estas funciones de personalización a medida que trabajemos con conjuntos de datos más interesantes a lo largo de este capítulo.

Uso de estilos integrados

Matplotlib tiene varios estilos predefinidos disponibles, con buenas configuraciones iniciales para colores de fondo, líneas de cuadrícula, anchos de línea, fuentes, tamaños de fuente y más que harán que sus visualizaciones sean atractivas sin requerir mucha personalización. Para ver los estilos disponibles en su sistema, ejecute las siguientes líneas en una sesión de terminal:

```
>>> importar matplotlib.pyplot como plt
>>> plt.style.disponible
['seaborn-dark', 'seaborn-darkgrid', 'seaborn-ticks', 'cinco y treinta y ocho',
--recorte--
```

Para usar cualquiera de estos estilos, agregue una línea de código antes de comenzar a generar la gráfica:

mpl_cuadrados.py

```
importar matplotlib.pyplot como plt

valores_de_entrada = [1, 2, 3, 4, 5]
cuadrados = [1, 4, 9, 16, 25]

plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
--recorte--
```

Este código genera el gráfico que se muestra en la figura 15-4. Hay disponible una amplia variedad de estilos; Juega con estos estilos para encontrar algunos que te gusten.

Figura 15-4: El estilo Seaborn incorporado

Trazado y estilo de puntos individuales con dispersión ()

A veces, es útil trazar y diseñar puntos individuales en función de ciertas características. Por ejemplo, puede trazar valores pequeños en un color y valores más grandes en un color diferente. También podría trazar un gran conjunto de datos con

un conjunto de opciones de estilo y luego enfatice puntos individuales al volver a trazarlos con diferentes opciones.

Para trazar un solo punto, use el método scatter() . Pasar el sencillo *(x,* y) valores del punto de interés para scatter() para trazar esos valores:

scatter_squares.py

```
importar matplotlib.pyplot como plt

plt.style.use('seaborn') fig, ax
= plt.subplots() ax.scatter(2, 4)


plt.mostrar()
```

Modifiquemos el resultado para hacerlo más interesante. Añadiremos un título, etiqueta los ejes y asegúrese de que todo el texto sea lo suficientemente grande para leer:

```
importar matplotlib.pyplot como plt

plt.style.use('seaborn') fig, ax
= plt.subplots() ÿ ax.scatter(2,
4, s=200)

# Establecer el título del gráfico y los ejes de etiquetas.
ax.set_title("Números cuadrados", tamaño de fuente=24)
ax.set_xlabel("Valor", tamaño de fuente=14)
ax.set_ylabel("Cuadrado del valor", tamaño de fuente=14)

# Establecer el tamaño de las etiquetas de marca.
ax.tick_params(axis='ambos', cual='principal', labelsize=14)

plt.mostrar()
```

En ÿ llamamos a scatter() y usamos el argumento s para establecer el tamaño de los puntos usados para dibujar el gráfico. Cuando ejecute *scatter_squares.py* ahora, debería ver un solo punto en el medio del gráfico, como se muestra en la Figura 15-5.

Figura 15-5: Trazado de un solo punto

Trazar una serie de puntos con dispersión ()

Para trazar una serie de puntos, podemos pasar scatter() listas separadas de valores x e y, así:

scatter_squares.py

```
importar matplotlib.pyplot como plt

valores_x = [1, 2, 3, 4, 5]
valores_y = [1, 4, 9, 16, 25]

plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
ax.scatter(valores_x, valores_y, s=100)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--
```

La lista x_values contiene los números que se van a elevar al cuadrado y y_values contiene el cuadrado de cada número. Cuando estas listas se pasan a scatter(), Matplotlib lee un valor de cada lista a medida que traza cada punto. Los puntos a trazar son (1, 1), (2, 4), (3, 9), (4, 16) y (5, 25); La figura 15-6 muestra el resultado.

Figura 15-6: Un diagrama de dispersión con múltiples puntos

Cálculo de datos automáticamente

Escribir listas a mano puede resultar ineficaz, sobre todo cuando tenemos muchos puntos. En lugar de pasar nuestros puntos en una lista, usemos un ciclo en Python para hacer los cálculos por nosotros.

Así es como se vería esto con 1000 puntos:

scatter_squares.py

```
importar matplotlib.pyplot como plt

ÿ valores_x = rango(1, 1001)
valores_y = [x**2 para x en valores_x]

plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
ÿ ax.scatter(valores_x, valores_y, s=10)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--

# Establecer el rango para cada eje.
ÿ eje.eje([0, 1100, 0, 1100000])

plt.mostrar()
```

Comenzamos con un rango de valores de x que contiene los números del 1 al 1000 ÿ. A continuación, una lista por comprensión genera los valores de y recorriendo los valores de x (para x en x_values), elevando al cuadrado cada número (x**2) y almacenando los resultados en y_values. Luego pasamos las listas de entrada y salida a scatter() ÿ. Debido a que este es un conjunto de datos grande, usamos un tamaño de punto más pequeño.

En ÿ usamos el método axis() para especificar el rango de cada eje. El método axis() requiere cuatro valores: los valores mínimo y máximo para el eje x y el eje y. Aquí, ejecutamos el eje x de 0 a 1100 y el eje y de 0 a 1,100,000. La figura 15-7 muestra el resultado.

Figura 15-7: Python puede trazar 1000 puntos tan fácilmente como traza 5 puntos.

Definición de colores personalizados

Para cambiar el color de los puntos, pase c a scatter() con el nombre de un color para usar entre comillas, como se muestra aquí:

```
ax.scatter(x_values, y_values, c='red', s=10)
```

También puede definir colores personalizados utilizando el modelo de color RGB. Definir un color, pase el argumento c una tupla con tres valores decimales (uno para rojo, verde y azul en ese orden), usando valores entre 0 y 1. Por ejemplo, la siguiente línea crea una gráfica con puntos de color verde claro:

```
ax.scatter(x_values, y_values, c=(0, 0.8, 0), s=10)
```

Los valores cercanos a 0 producen colores oscuros y los valores cercanos a 1 producen colores más claros.

Usando un mapa de colores

Un *mapa* de colores es una serie de colores en un degradado que se mueve desde un color inicial hasta uno final. Los mapas de colores se utilizan en las visualizaciones para enfatizar un patrón en los datos. Por ejemplo, puede convertir los valores bajos en un color claro y los valores altos en un color más oscuro.

El módulo pyplot incluye un conjunto de mapas de colores integrados. Para usar uno de estos mapas de colores, debe especificar cómo pyplot debe asignar un color a cada punto en el conjunto de datos. Aquí se explica cómo asignar un color a cada punto en función de su valor y:

scatter_squares.py

```
importar matplotlib.pyplot como plt

valores_x = rango (1, 1001)
valores_y = [x**2 para x en valores_x]

ax.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)

# Establecer el título del gráfico y los ejes de etiquetas.
--recorte--
```

Pasamos la lista de valores y a c, y luego le decimos a pyplot qué mapa de colores usar usando el argumento cmap . Este código colorea los puntos con valores de y más bajos de azul claro y los puntos con valores de y más altos de azul oscuro. La figura 15-8 muestra el gráfico resultante.

Puede ver todos los mapas de colores disponibles en pyplot en https://matplotlib.org/; *ir a* ***Ejemplos,*** *desplácese hacia abajo hasta* ***Color*** *y haga clic en* ***Referencia de mapa de colores.***

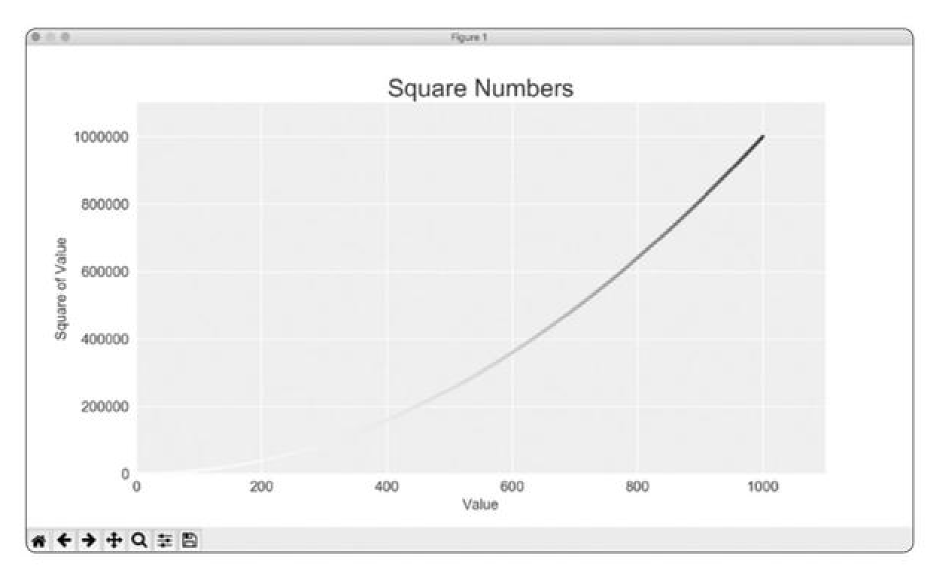


Figura 15-8: Un gráfico usando el mapa de colores *Blues*

Guardar sus parcelas automáticamente

Si desea que su programa guarde automáticamente el gráfico en un archivo, puede reemplazar la llamada a plt.show() con una llamada a plt.savefig():

```
plt.savefig('cuadrados_plot.png', bbox_inches='apretado')
```

El primer argumento es un nombre de archivo para la imagen de la trama, que se guardará en el mismo directorio que *scatter_squares.py.* El segundo argumento recorta los espacios en blanco adicionales de la trama. Si desea el espacio en blanco adicional alrededor de la trama, simplemente omita este argumento.

Inténtalo tú mismo

15-1. Cubos: Un número elevado a la tercera potencia es un cubo. Traza los primeros cinco números cúbicos y luego traza los primeros 5000 números cúbicos.

15-2. Cubos de colores: aplique un mapa de colores a su diagrama de cubos.

Paseos aleatorios

En esta sección, usaremos Python para generar datos para una caminata aleatoria y luego usaremos Matplotlib para crear una representación visualmente atractiva de esos datos. Una *caminata aleatoria* es un camino que no tiene una dirección clara pero que está determinado por una serie de decisiones aleatorias, cada una de las cuales se deja completamente al azar. Podrías imaginar una caminata aleatoria como el camino que tomaría una hormiga confundida si diera cada paso en una dirección aleatoria.

Los paseos aleatorios tienen aplicaciones prácticas en la naturaleza, la física, la biología, la química y la economía. Por ejemplo, un grano de polen que flota en una gota de agua se mueve por la superficie del agua porque las moléculas de agua lo empujan constantemente. El movimiento molecular en una gota de agua es aleatorio, por lo que el camino que traza un grano de polen en la superficie es un camino aleatorio.

El código que escribiremos a continuación modela muchas situaciones del mundo real.

Crear la clase RandomWalk()

Para crear una caminata aleatoria, crearemos una clase RandomWalk , que tomará decisiones aleatorias sobre la dirección que debe tomar la caminata. La clase necesita tres atributos: una variable para almacenar el número de puntos del recorrido y dos listas para almacenar los valores de las coordenadas x e y de cada punto del recorrido.

Solo necesitaremos dos métodos para la clase RandomWalk : el __init__() y fill_walk(), que calculará los puntos en la caminata. Comencemos con __init__() como se muestra aquí:

paseo_aleatorio.py

```
ÿ de elección de importación aleatoria

Clase Paseo Aleatorio:
    """Una clase para generar paseos aleatorios."""

ÿ def __init__(self, num_points=5000):
        """Inicializar atributos de un paseo."""
        self.num_puntos = numero_puntos

        # Todos los recorridos comienzan en (0, 0).
ÿ       self.x_values = [0]
        self.y_valores = [0]
```

Para tomar decisiones aleatorias, almacenaremos los posibles movimientos en una lista y usaremos la función choice() , del módulo aleatorio , para decidir qué movimiento hacer cada vez que se da un paso ÿ. Luego establecemos el número predeterminado de puntos en una caminata en 5000, que es lo suficientemente grande como para generar algunos patrones interesantes pero lo suficientemente pequeño como para generar caminatas rápidamente ÿ. Luego, en ÿ, hacemos dos listas para contener los valores de x e y, y comenzamos cada caminata en el punto (0, 0).

Elegir direcciones

Usaremos el método fill_walk() , como se muestra aquí, para llenar nuestra caminata con puntos y determinar la dirección de cada paso. Agregue este método a *random_walk.py:*

paseo_aleatorio.py

```
    def fill_walk(auto):
        """Calcular todos los puntos del paseo."""

        # Siga dando pasos hasta que la caminata alcance la longitud deseada.
ÿ       while len(self.x_values) < self.num_points:

            # Decide en qué dirección ir y qué tan lejos ir en esa dirección.
ÿ           dirección_x = opción ([1, -1])
```

```
            x_distancia = elección ([0, 1, 2, 3, 4])
ÿ           x_paso = x_dirección * x_distancia

            dirección_y = elección([1, -1])
            distancia_y = elección([0, 1, 2, 3, 4])
ÿ           paso_y = dirección_y * distancia_y

            # Rechazar movimientos que no van a ninguna parte.
ÿ           si x_paso == 0 y y_paso == 0:
                Seguir

            # Calcular la nueva posición.
ÿ           x = self.x_values[-1] + x_step
            y = self.y_values[-1] + y_step

            self.x_values.append(x)
            self.y_values.append(y)
```

En ÿ configuramos un bucle que se ejecuta hasta que la caminata se llena con el correcto número de puntos. La parte principal del método fill_walk() le dice a Python cómo simular cuatro decisiones aleatorias: ¿la caminata irá a la derecha o a la izquierda? ¿Hasta dónde llegará en esa dirección? ¿Subirá o bajará? ¿Hasta dónde llegará en esa dirección?

Usamos choice([1, -1]) para elegir un valor para x_direction, que devuelve 1 para el movimiento a la derecha o ÿ1 para la izquierda ÿ. A continuación, elección ([0, 1, 2, 3, 4]) le dice a Python qué tan lejos debe moverse en esa dirección (x_distance) seleccionando aleatoriamente un número entero entre 0 y 4. (La inclusión de un 0 nos permite dar pasos a lo largo del eje y, así como también pasos que tienen movimiento a lo largo de ambos ejes).

En ÿ y ÿ determinamos la longitud de cada paso en las direcciones *x* e *y* multiplicando la dirección del movimiento por la distancia elegida. Un resultado positivo para x_step significa moverse a la derecha, un resultado negativo significa moverse a la izquierda y 0 significa moverse verticalmente. Un resultado positivo para y_step significa moverse hacia arriba, negativo significa moverse hacia abajo y 0 significa moverse horizontalmente. Si el valor de x_step y y_step es 0, el paseo no va a ninguna parte, así que continuamos el ciclo para ignorar este movimiento ÿ.

Para obtener el siguiente valor de x para la caminata, sumamos el valor en x_step al último valor almacenado en x_values ÿ y hacemos lo mismo para los valores de y. Cuando tenemos estos valores, los agregamos a x_values y y_values.

Trazar el paseo aleatorio

Aquí está el código para trazar todos los puntos en la caminata:

rw_visual.py

```
importar matplotlib.pyplot como plt

de random_walk importar RandomWalk

# Haz un paseo aleatorio.
ÿ rw = Paseo aleatorio()
rw.llenar_caminar()
```

```
# Trace los puntos en la caminata.
plt.style.use('clásico')
higo, hacha = plt.subplots()
ÿ ax.scatter(rw.x_values, rw.y_values, s=15)
plt.mostrar()
```

Comenzamos importando pyplot y RandomWalk. Luego creamos una caminata aleatoria y la almacenamos en rw ÿ, asegurándonos de llamar a fill_walk(). En ÿ alimentamos los valores x e y de la caminata para scatter() y elegimos un tamaño de punto apropiado. La figura 15-9 muestra el gráfico resultante con 5000 puntos. (Las imágenes de esta sección omiten el visor de Matplotlib, pero seguirá viéndolo cuando ejecute *rw_visual.py).*

Figura 15-9: Un paseo aleatorio con 5000 puntos

Generación de múltiples recorridos aleatorios

Cada recorrido aleatorio es diferente y es divertido explorar los diversos patrones que se pueden generar. Una forma de utilizar el código anterior para realizar varios paseos sin tener que ejecutar el programa varias veces es envolverlo en un ciclo while , como este:

rw_visual.py

```
importar matplotlib.pyplot como plt

de random_walk importar RandomWalk

# Seguir realizando nuevos recorridos, siempre y cuando el programa esté activo.
mientras que es cierto:
    # Haz un paseo aleatorio.
    rw = Paseo Aleatorio()
    rw.llenar_caminar()

    # Trace los puntos en la caminata.
    plt.style.use('clásico')
```

```
    higo, hacha = plt.subplots()
    hacha.dispersión(rw.x_valores, rw.y_valores, s=15)
    plt.mostrar()

    keep_running = input("¿Hacer otra caminata? (s/n): ")
    si sigue_ejecutando == 'n':
        romper
```

Este código genera un recorrido aleatorio, lo muestra en el visor de Matplotlib y se detiene con el visor abierto. Cuando cierre el visor, se le preguntará si desea generar otra caminata. Presione **y** para generar caminos que permanezcan cerca del punto de partida, que se desvíen principalmente en una dirección, que tengan secciones delgadas que conecten grupos más grandes de puntos, etc. Cuando desee finalizar el programa, presione n.

Estilo de la caminata

En esta sección, personalizaremos nuestras tramas para enfatizar las características importantes de cada caminata y restar énfasis a los elementos que distraen. Para ello, identificamos las características que queremos destacar, como por ejemplo dónde empezó la caminata, dónde terminó y el camino recorrido. A continuación, identificamos las características para quitar énfasis, como las marcas y las etiquetas. El resultado debe ser una representación visual simple que comunique claramente el camino tomado en cada recorrido aleatorio.

Colorear los puntos

Usaremos un mapa de colores para mostrar el orden de los puntos en el recorrido y luego eliminaremos el contorno negro de cada punto para que el color de los puntos sea más claro. Para colorear los puntos según su posición en el paseo, le pasamos al argumento c una lista que contiene la posición de cada punto. Debido a que los puntos se trazan en orden, la lista solo contiene los números del 0 al 4999, como se muestra aquí:

rw_visual.py

```
--recorte--
mientras que es cierto:
    # Haz un paseo aleatorio.
    rw = Paseo Aleatorio()
    rw.llenar_caminar()

    # Trace los puntos en la caminata.
    plt.style.use('clásico')
    higo, hacha = plt.subplots()
ÿ numero_puntos = rango(rw.num_puntos)
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
        edgecolors='ninguno', s=15)
    plt.mostrar()

    keep_running = input("¿Hacer otra caminata? (s/n): ")
    --recorte--
```

En ÿ usamos range() para generar una lista de números iguales al número de puntos en el paseo. Luego los almacenamos en la lista point_numbers, que usaremos para establecer el color de cada punto en la caminata. Pasamos point_numbers al argumento c , use el mapa de colores Blues y luego pase edgecolors='none' para deshacerse del contorno negro alrededor de cada punto. El resultado es un gráfico de la caminata que varía de azul claro a azul oscuro a lo largo de un gradiente, como se muestra en la Figura 15-10.

Figura 15-10: Un recorrido aleatorio coloreado con el mapa de colores *Blues*

Trazado de los puntos de inicio y finalización

Además de colorear los puntos para mostrar su posición a lo largo de la caminata, sería útil ver dónde comienza y termina cada caminata. Para hacerlo, podemos graficar el primer y el último punto individualmente después de que se haya graficado la serie principal. Haremos los puntos finales más grandes y los colorearemos de manera diferente para que se destaquen, como se muestra aquí:

rw_visual.py

```
--recorte--
mientras que es cierto:
    --recorte--
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
        edgecolors='ninguno', s=15)

    # Enfatice los puntos primero y último.
    ax.scatter(0, 0, c='verde', edgecolors='ninguno', s=100)
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
        s = 100)

    plt.mostrar()
    --recorte--
```

Para mostrar el punto de partida, trazamos el punto (0, 0) en verde en un tamaño mayor (s=100) que el resto de los puntos. Para marcar el punto final, trazamos el último valor x e y en la caminata en rojo con un tamaño de 100. Asegúrese de insertar este código justo antes de la llamada a plt.show() para que los puntos inicial y final sean dibujado encima de todos los otros puntos.

Cuando ejecute este código, debería poder detectar exactamente dónde se encuentra cada caminata comienza y termina. (Si estos puntos finales no se destacan claramente, ajuste su color y tamaño hasta que lo hagan).

Limpiando las hachas

Eliminemos los ejes en este diagrama para que no distraigan la atención del camino de cada caminata. Para apagar los ejes, use este código:

rw_visual.py

```
--recorte--
mientras que es cierto:
    --recorte--
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
        s = 100)

    # Retire los ejes.
ÿ   ax.get_xaxis().set_visible(Falso)
    ax.get_yaxis().set_visible(Falso)

    plt.mostrar()
    --recorte--
```

Para modificar los ejes, usamos los métodos ax.get_xaxis() y ax.get_yaxis() ÿ para establecer la visibilidad de cada eje en False. A medida que continúe trabajando con visualizaciones, verá con frecuencia este encadenamiento de métodos.

Ejecute *rw_visual.py* ahora; deberías ver una serie de gráficos sin ejes.

Adición de puntos de trazado

Aumentemos el número de puntos para darnos más datos con los que trabajar. Para hacerlo, aumentamos el valor de num_points cuando hacemos un RandomWalk instancia y ajuste el tamaño de cada punto al dibujar el gráfico, como se muestra aquí:

rw_visual.py

```
--recorte--
mientras que es cierto:
    # Haz un paseo aleatorio.
    rw = paseo aleatorio (50_000)
    rw.llenar_caminar()

    # Trace los puntos en la caminata.
    plt.style.use('clásico')
    higo, hacha = plt.subplots()
    numeros_puntos = rango(rw.num_puntos)
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
        edgecolor='ninguno', s=1)
    --recorte--
```

Este ejemplo crea un recorrido aleatorio con 50 000 puntos (para reflejar datos del mundo real) y traza cada punto con un tamaño s=1. La caminata resultante es tenue y parecida a una nube, como se muestra en la Figura 15-11. Como puede ver, ¡hemos creado una obra de arte a partir de un diagrama de dispersión simple!

Experimente con este código para ver cuánto puede aumentar el número de puntos en una caminata antes de que su sistema comience a ralentizarse significativamente o la trama pierda su atractivo visual.

Figura 15-11: Un paseo con 50.000 puntos

Alterar el tamaño para llenar la pantalla

Una visualización es mucho más eficaz para comunicar patrones en los datos si encaja bien en la pantalla. Para que la ventana de trazado se ajuste mejor a su pantalla, ajuste el tamaño de la salida de Matplotlib, así:

rw_visual.py

```
--recorte--
mientras que es cierto:
    # Haz un paseo aleatorio.
    rw = paseo aleatorio (50_000)
    rw.llenar_caminar()

    # Trace los puntos en la caminata.
    plt.style.use('clásico')
    higo, hacha = plt.subplots(figsize=(15, 9))
    --recorte--
```

Al crear la trama, puede pasar un argumento figsize para establecer el tamaño de la figura. El parámetro figsize toma una tupla, que le dice a Matplotlib las dimensiones de la ventana de trazado en pulgadas.

Matplotlib asume que la resolución de su pantalla es de 100 píxeles por pulgada; si este código no le da un tamaño de gráfico preciso, ajuste los números como

necesario. O, si conoce la resolución de su sistema, pase plt.subplots() la resolución utilizando el parámetro dpi para establecer un tamaño de gráfico que haga un uso efectivo del espacio disponible en su pantalla, como se muestra aquí:

```
higo, hacha = plt.subplots(figsize=(10, 6), dpi=128)
```

Inténtalo tú mismo

15-3. Movimiento molecular: modifique rw_visual.py reemplazando plt.scatter() con plt.plot(). Para simular la ruta de un grano de polen en la superficie de una gota de agua, pase los valores rw.x_values y rw.y_values e incluya un argumento de ancho de línea . Use 5000 en lugar de 50,000 puntos.

15-4. Paseos aleatorios modificados: en la clase RandomWalk , x_step y y_step se generan a partir del mismo conjunto de condiciones. La dirección se elige aleatoriamente de la lista [1, -1] y la distancia de la lista [0, 1, 2, 3, 4]. Modifique los valores en estas listas para ver qué sucede con la forma general de sus caminatas. Pruebe con una lista más larga de opciones para la distancia, como 0 a 8, o elimine el ÿ1 de la lista de direcciones x o y.

15-5. Refactorización: el método fill_walk() es largo. Cree un nuevo método llamado get_step() para determinar la dirección y la distancia de cada paso y luego calcule el paso. Debería terminar con dos llamadas a get_step() en fill_walk():

```
x_paso = self.obtener_paso()
y_paso = self.obtener_paso()
```

Esta refactorización debería reducir el tamaño de fill_walk() y hacer que el método sea más fácil de leer y comprender.

Tirar dados con Plotly

En esta sección, usaremos el paquete de Python Plotly para producir visualizaciones interactivas. Plotly es particularmente útil cuando crea visualizaciones que se mostrarán en un navegador, porque las visualizaciones se escalarán automáticamente para adaptarse a la pantalla del espectador. Las visualizaciones que genera Plotly también son interactivas; cuando el usuario se desplaza sobre ciertos elementos en la pantalla, se resalta la información sobre ese elemento.

En este proyecto, analizaremos los resultados de lanzar dados. Cuando lanzas un dado normal de seis caras, tienes las mismas posibilidades de obtener cualquiera de los números del 1 al 6. Sin embargo, cuando usas dos dados, es más probable que obtengas ciertos números en lugar de otros. Trataremos de determinar

qué números es más probable que ocurran al generar un conjunto de datos que represente el lanzamiento de dados. Luego trazaremos los resultados de una gran cantidad de tiradas para determinar qué resultados son más probables que otros.

El estudio de tirar los dados se usa a menudo en matemáticas para explicar varios tipos de análisis de datos. Pero también tiene aplicaciones del mundo real en casinos y otros escenarios de juego, así como en la forma en que se juegan juegos como Monopoly y muchos juegos de rol.

Instalación de plotly

Instale Plotly usando pip, tal como lo hizo con Matplotlib:

```
$ python -m pip install --usuario plotly
```

Si usó **python3** o algo más al instalar Matplotlib, asegúrese de usar el mismo comando aquí.

Para ver qué tipo de visualizaciones son posibles con Plotly, visite la galería de tipos de gráficos en *https:// plot.ly/ python/.* Cada ejemplo incluye código fuente, para que pueda ver cómo Plotly genera las visualizaciones.

Creación de la clase de troquel

Crearemos la siguiente clase Die para simular el lanzamiento de un dado:

morir.py

```
de randint de importación aleatoria

clase morir:
    """Una clase que representa un solo dado."""

ÿ def __init__(self, num_sides=6):
        """Supongamos un dado de seis caras."""
        self.num_lados = numero_lados

    rollo def (uno mismo):
        """"Retorna un valor aleatorio entre 1 y el número de lados."""
ÿ       return randint(1, self.num_sides)
```

El método __init__() toma un argumento opcional. con el dado clase, cuando se crea una instancia de nuestro dado, el número de lados siempre será seis si no se incluye ningún argumento. Si *se* incluye un argumento , ese valor establecerá el número de caras del dado ÿ. (Los dados se nombran por su número de lados: un dado de seis lados es un D6, un dado de ocho lados es un D8, y así sucesivamente).

El método roll() utiliza la función randint() para devolver un número aleatorio entre 1 y el número de lados ÿ. Esta función puede devolver el valor inicial (1), el valor final (num_sides) o cualquier número entero entre los dos.

rodar el dado

Antes de crear una visualización basada en la clase Die , tiremos un D6, imprimamos los resultados y verifiquemos que los resultados parezcan razonables:

die_visual.py

```
de morir importar morir

# Crea un D6.
ÿ morir = morir()

# Haga algunas tiradas y almacene los resultados en una lista.
resultados = []
ÿ para roll_num en rango (100):
    resultado = morir.tirar()
    resultados.append(resultado)

imprimir (resultados)
```

En ÿ creamos una instancia de Die con los seis lados predeterminados. En ÿ rodamos el dado 100 veces y almacena los resultados de cada tirada en la lista de resultados. Aquí hay un conjunto de muestra de resultados:

```
[4, 6, 5, 6, 1, 5, 6, 3, 5, 3, 5, 3, 2, 2, 1, 3, 1, 5, 3, 6, 3, 6, 5, 4,
 1, 1, 4, 2, 3, 6, 4, 2, 6, 4, 1, 3, 2, 5, 6, 3, 6, 2, 1, 1, 3, 4, 1, 4,
 3, 5, 1, 4, 5, 5, 2, 3, 3, 1, 2, 3, 5, 6, 2, 5, 6, 1, 3, 2, 1, 1, 1, 6,
 5, 5, 2, 2, 6, 4, 1, 4, 5, 1, 1, 1, 4, 5, 3, 3, 1, 3, 5, 4, 5, 6, 5, 4,
 1, 5, 1, 2]
```

Una exploración rápida de estos resultados muestra que la clase Die parece estar funcionando. Vemos los valores 1 y 6, por lo que sabemos que se están devolviendo los valores más pequeños y más grandes posibles, y como no vemos 0 o 7, sabemos que todos los resultados están en el rango apropiado. También vemos cada número del 1 al 6, lo que indica que todos los resultados posibles están representados.

Determinemos exactamente cuántas veces aparece cada número.

Analizando los resultados

Analizaremos los resultados de tirar un D6 contando cuántas veces tiramos cada número:

die_visual.py

```
--recorte--
# Haga algunas tiradas y almacene los resultados en una lista.
resultados = []
ÿ para roll_num en rango (1000):
    resultado = morir.tirar()
    resultados.append(resultado)

# Analizar los resultados.
frecuencias = []
ÿ para valor en rango (1, die.num_sides+1):
ÿ frecuencia = resultados.recuento(valor)
```

```
ÿ frecuencias.append(frecuencia)

  imprimir (frecuencias)
```

Debido a que ya no imprimimos los resultados, podemos aumentar el número de tiradas simuladas a 1000 ÿ. Para analizar las tiradas, creamos la lista vacía de frecuencias para almacenar el número de veces que se tira cada valor. Recorremos los valores posibles (del 1 al 6 en este caso) en ÿ, contamos cuántas veces aparece cada número en los resultados ÿ y luego agregamos este valor a la lista de frecuencias ÿ. Luego imprimimos esta lista antes de hacer una visualización:

```
[155, 167, 168, 170, 159, 181]
```

Estos resultados parecen razonables: vemos seis frecuencias, una para cada pos. número posible cuando lanzas un D6, y vemos que ninguna frecuencia es significativamente más alta que otra. Ahora vamos a visualizar estos resultados.

Hacer un histograma

Con una lista de frecuencias, podemos hacer un *histograma* de los resultados. Un histograma es un gráfico de barras que muestra la frecuencia con la que se producen determinados resultados. Aquí está el código para crear el histograma:

die_visual.py

```
de plotly.graph_objs barra de importación, diseño
desde plotly importar sin conexión

de morir importar morir
--recorte--

# Analizar los resultados.
frecuencias = []
para el valor en el rango (1, die.num_sides+1):
    frecuencia = resultados.recuento(valor)
    frecuencias.append(frecuencia)

# Visualiza los resultados.
ÿ valores_x = lista(rango(1, dado.num_lados+1))
ÿ datos = [Bar(x=x_valores, y=frecuencias)]

ÿ x_axis_config = {'título': 'Resultado'}
  y_axis_config = {'title': 'Frecuencia del resultado'}
ÿ my_layout = Layout(title='Resultados de tirar un D6 1000 veces',
        xaxis=x_axis_config, yaxis=y_axis_config)
ÿ offline.plot({'data': data, 'layout': my_layout}, filename='d6.html')
```

Para hacer un histograma, necesitamos una barra para cada uno de los posibles resultados. Los almacenamos en una lista llamada x_values, que comienza en 1 y termina en el número de lados del dado ÿ. Plotly no acepta los resultados del rango () directamente, por lo que necesitamos convertir el rango en una lista explícitamente usando

la función lista() . La clase Bar() de Plotly representa un conjunto de datos que se formateará como un gráfico de barras ÿ. Esta clase necesita una lista de valores de x y una lista de valores de y. La clase debe estar entre corchetes porque un conjunto de datos puede tener varios elementos.

Cada eje se puede configurar de varias maneras, y cada configuración opción de opción se almacena como una entrada en un diccionario. En este punto, solo estamos configurando el título de cada eje ÿ. La clase Layout() devuelve un objeto que especifica el diseño y la configuración del gráfico como un todo ÿ. Aquí establecemos el título del gráfico y también pasamos los diccionarios de configuración de los ejes x e y.

Para generar el gráfico, llamamos a la función offline.plot() ÿ. Esta función necesita un diccionario que contenga los datos y los objetos de diseño, y también acepta un nombre para el archivo donde se guardará el gráfico. Almacenamos la salida en un archivo llamado *d6.html.*

Cuando ejecute el programa *die_visual.py,* probablemente se abrirá un navegador que mostrará el archivo *d6.html.* Si esto no sucede automáticamente, abra una nueva pestaña en cualquier navegador web y luego abra el archivo *d6.html* (en la carpeta donde guardó *die_visual.py).* Debería ver un gráfico similar al de la Figura 15-12. (He modificado ligeramente este gráfico para imprimirlo; de manera predeterminada, Plotly genera gráficos con texto más pequeño que el que ve aquí).

Figura 15-12: Un gráfico de barras simple creado con Plotly

Tenga en cuenta que Plotly ha hecho que el gráfico sea interactivo: desplace el cursor sobre cualquier barra del gráfico y verá los datos asociados. Esta función es especialmente útil cuando se trazan varios conjuntos de datos en el mismo gráfico. Observe también los íconos en la parte superior derecha, que le permiten desplazar y hacer zoom en la visualización, y guardar su visualización como una imagen.

rodar dos dados

Tirar dos dados da como resultado números más grandes y una distribución diferente de los resultados. Modifiquemos nuestro código para crear dos dados D6 para simular la forma en que lanzamos un par de dados. Cada vez que lancemos el par, sumaremos los dos números (uno de cada dado) y almacenaremos la suma en los resultados. Guarde una copia de *die_visual.py* como *dice_visual.py,* y realice los siguientes cambios:

dados_visual.py

```
de plotly.graph_objs barra de importación, diseño
desde plotly importar sin conexión

de morir importar morir

# Crea dos dados D6.
morir_1 = morir()
morir_2 = morir()

# Haga algunas tiradas y almacene los resultados en una lista.
resultados = []
para roll_num en rango (1000):
ÿ resultado = dado_1.tirar() + dado_2.tirar()
        resultados.append(resultado)

# Analizar los resultados.
frecuencias = []
ÿ max_result = dado_1.num_lados + dado_2.num_lados
ÿ para valor en rango (2, max_result+1):
        frecuencia = resultados.recuento(valor)
        frecuencias.append(frecuencia)

# Visualiza los resultados.
x_values = list(range(2, max_result+1))
datos = [Bar(x=x_valores, y=frecuencias)]

ÿ x_axis_config = {'título': 'Resultado', 'dtick': 1}
y_axis_config = {'title': 'Frecuencia del resultado'}
my_layout = Layout(title='Resultados de tirar dos dados D6 1000 veces',
            xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6_d6.html')
```

Después de crear dos instancias de Die, tiramos los dados y calculamos la suma de los dos dados para cada tirada ÿ. El mayor resultado posible (12) es la suma del mayor número de ambos dados, que almacenamos en max_result ÿ.
El resultado más pequeño posible (2) es la suma del número más pequeño en ambos dados. Cuando analizamos los resultados, contamos el número de resultados para cada valor entre 2 y max_result ÿ. (Podríamos haber usado el rango (2, 13), pero esto funcionaría solo para dos dados D6. Al modelar situaciones del mundo real, es mejor escribir código que pueda modelar fácilmente una variedad de situaciones. Este código nos permite simular el lanzamiento un par de dados con cualquier número de lados).

Al crear el gráfico, incluimos la tecla dtick en x_axis_config diccionario ÿ. Esta configuración controla el espacio entre las marcas en el eje x. Ahora que tenemos más barras en el histograma, el valor predeterminado de Plotly

la configuración solo etiquetará algunas de las barras. La configuración 'dtick': 1 le dice a Plotly que etiquete cada marca de verificación. También actualizamos el título del gráfico y también cambiamos el nombre del archivo de salida.

Después de ejecutar este código, debería ver un gráfico similar al de la Figura 15-13.

Figura 15-13: Resultados simulados de lanzar dos dados de seis caras 1000 veces

Este gráfico muestra los resultados aproximados que es probable que obtenga cuando tira un par de dados D6. Como puede ver, es menos probable que saque un 2 o un 12 y lo más probable es que saque un 7. Esto sucede porque hay seis formas de sacar un 7, a saber: 1 y 6, 2 y 5, 3 y 4 , 4 y 3, 5 y 2, o 6 y 1.

Dados rodantes de diferentes tamaños

Vamos a crear un dado de seis lados y un dado de diez lados, y veamos qué sucede cuando los lanzamos 50,000 veces:

dados_visual.py

```
de plotly.graph_objs barra de importación, diseño
desde plotly importar sin conexión

de morir importar morir

# Crea un D6 y un D10.
morir_1 = morir()
ÿ dado_2 = Dado(10)

# Haga algunas tiradas y almacene los resultados en una lista.
resultados = []
para roll_num en rango (50_000):
    resultado = dado_1.tirar() + dado_2.tirar()
    resultados.append(resultado)
```

```
# Analizar los resultados.
--recorte--

# Visualiza los resultados.
x_values = list(range(2, max_result+1))
datos = [Bar(x=x_valores, y=frecuencias)]

x_axis_config = {'título': 'Resultado', 'dtick': 1}
y_axis_config = {'title': 'Frecuencia del resultado'}
ÿ my_layout = Layout(title='Resultados de tirar un D6 y un D10 50000 veces',
        xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6_d10.html')
```

Para hacer un D10, pasamos el argumento 10 al crear la segunda instancia de Die ÿ y cambiamos el primer bucle para simular 50 000 rollos en lugar de 1000. Cambiamos el título del gráfico y actualizamos el nombre del archivo de salida también ÿ.

La figura 15-14 muestra el gráfico resultante. En lugar de un resultado más probable, hay cinco. Esto sucede porque todavía hay una sola forma de obtener el valor más pequeño (1 y 1) y el valor más grande (6 y 10), pero el dado más pequeño limita la cantidad de formas en que puede generar los números del medio: hay seis formas de obtener un 7, 8, 9, 10 y 11. Por lo tanto, estos son los resultados más comunes y es igualmente probable que obtenga cualquiera de estos números.

Figura 15-14: Los resultados de lanzar un dado de seis caras y uno de diez caras 50 000 veces

Nuestra capacidad de usar Plotly para modelar el lanzamiento de dados nos da una libertad considerable para explorar este fenómeno. En solo unos minutos puedes simular una gran cantidad de tiradas usando una gran variedad de dados.

Inténtalo tú mismo

15-6. Dos D8: crea una simulación que muestre lo que sucede cuando lanzas dos dados de ocho caras 1000 veces. Trate de imaginar cómo cree que se verá la visualización antes de ejecutar la simulación; luego vea si su intuición era correcta.
Aumente gradualmente la cantidad de rollos hasta que comience a ver los límites de las capacidades de su sistema.

15-7. Tres dados: cuando lanzas tres dados D6, el número más pequeño que puedes lanzar es 3 y el número más grande es 18. Crea una visualización que muestre lo que sucede cuando lanzas tres dados D6.

15-8. Multiplicación: cuando lanzas dos dados, generalmente sumas los dos números para obtener el resultado. Cree una visualización que muestre lo que sucede si multiplica estos números.

15-9. Comprensiones de matrices: para mayor claridad, los listados en esta sección usan la forma larga de bucles for . Si se siente cómodo usando listas de comprensión, intente escribir una comprensión para uno o ambos bucles en cada uno de estos programas.

15-10. Practicar con ambas bibliotecas: intente usar Matplotlib para hacer una visualización de tirada de dados y use Plotly para hacer la visualización de una caminata aleatoria. (Deberá consultar la documentación de cada biblioteca para completar este ejercicio).

Resumen

En este capítulo, aprendió a generar conjuntos de datos y crear visualizaciones de esos datos. Creó gráficos simples con Matplotlib y usó un gráfico de dispersión para explorar recorridos aleatorios. También creó un histograma con Plotly y usó un histograma para explorar los resultados de lanzar dados de diferentes tamaños.

Generar sus propios conjuntos de datos con código es una forma interesante y poderosa de modelar y explorar una amplia variedad de situaciones del mundo real. A medida que continúe trabajando en los proyectos de visualización de datos que siguen, esté atento a las situaciones que podría modelar con código. Mire las visualizaciones que ve en los medios de comunicación y vea si puede identificar aquellas que se generaron usando métodos similares a los que está aprendiendo en estos proyectos.

En el Capítulo 16, descargará datos de fuentes en línea y continuará usar Matplotlib y Plotly para explorar esos datos.



dieciséis

Descarga de datos

En este capítulo, descargará conjuntos de datos de fuentes en línea y creará visualizaciones funcionales de esos datos. Puedes encontrar increíble variedad de datos en línea, muchos de los cuales no han sido examinados a fondo. La capacidad de analizar estos datos le permite descubrir patrones y conexiones que nadie más ha encontrado.

Accederemos y visualizaremos los datos almacenados en dos formatos de datos comunes, CSV y JSON. Usaremos el módulo csv de Python para procesar los datos meteorológicos almacenados en el formato CSV (valores separados por comas) y analizaremos las temperaturas altas y bajas a lo largo del tiempo en dos ubicaciones diferentes. Luego usaremos Matplotlib para generar un gráfico basado en nuestros datos descargados para mostrar las variaciones de temperatura en dos entornos diferentes: Sitka, Alaska y Death Valley, California. Más adelante en el capítulo, usaremos el módulo json para acceder a los datos de terremotos almacenados en formato JSON y usaremos Plotly para dibujar un mapa mundial que muestre las ubicaciones y las magnitudes de los terremotos recientes.

Al final de este capítulo, estará preparado para trabajar con diferentes tipos y formatos de conjuntos de datos, y tendrá una comprensión más profunda de cómo crear visualizaciones complejas. Poder acceder y visualizar datos en línea de diferentes tipos y formatos es esencial para trabajar con una amplia variedad de conjuntos de datos del mundo real.

El formato de archivo CSV

Una forma sencilla de almacenar datos en un archivo de texto es escribir los datos como una serie de valores separados por comas, lo que se denomina valores separados por comas . Los archivos resultantes se denominan archivos *CSV .* Por ejemplo, aquí hay una porción de datos meteorológicos en formato CSV:

```
"USW00025333","SITKA AIRPORT, AK US","2018-01-01","0.45","48","38"
```

Este es un extracto de algunos datos meteorológicos del 1 de enero de 2018 en Sitka, Alaska. Incluye las temperaturas máximas y mínimas del día, así como una serie de otras mediciones de ese día. Los archivos CSV pueden ser difíciles de leer para los humanos, pero son fáciles de procesar y extraer valores para los programas, lo que acelera el proceso de análisis de datos.

Comenzaremos con un pequeño conjunto de datos meteorológicos en formato CSV registrados en Sitka, que está disponible en los recursos del libro en *https:// nostarch.com / pythoncrashcourse2e/.* Cree una carpeta llamada *datos* dentro de la carpeta donde está guardando los programas de este capítulo. Copie el archivo *sitka_weather_07-2018 _simple.csv* en esta nueva carpeta. (Después de descargar los recursos del libro, tendrá todos los archivos que necesita para este proyecto).

Los datos meteorológicos de este proyecto se descargaron originalmente de https://ncdc .noaa.gov/cdo-web/.

Análisis de los encabezados del archivo CSV

El módulo csv de Python en la biblioteca estándar analiza las líneas en un archivo CSV y nos permite extraer rápidamente los valores que nos interesan. Comencemos examinando la primera línea del archivo, que contiene una serie de encabezados para los datos. Estos encabezados nos dicen qué tipo de información contienen los datos:

sitka_highs.py

```
importar csv

nombre de archivo = 'datos/sitka_weather_07-2018_simple.csv'
u con abierto (nombre de archivo) como f:
v lector = csv.lector(f)
w header_row = siguiente (lector)
    imprimir (fila_encabezado)
```

Después de importar el módulo csv , asignamos el nombre del archivo con el que estamos trabajando a filename. Luego abrimos el archivo y asignamos el archivo resultante

objeto a f u. A continuación, llamamos a csv.reader() y le pasamos el objeto de archivo como argumento para crear un objeto de lector asociado con ese archivo v. Asignamos el objeto de lector a lector.

El módulo csv contiene una función next() , que devuelve la siguiente línea en el archivo cuando pasa el objeto del lector. En la lista anterior, llamamos a next() solo una vez para obtener la primera línea del archivo, que contiene los encabezados de archivo w. Almacenamos los datos que se devuelven en header_row. Como puede ver, header_row contiene encabezados significativos relacionados con el clima que nos dicen qué información contiene cada línea de datos:

```
['ESTACIÓN', 'NOMBRE', 'FECHA', 'PRCP', 'TAVG', 'TMAX', 'TMIN']
```

El objeto lector procesa la primera línea de valores separados por comas en el archivo y almacena cada uno como un elemento en una lista. El encabezado ESTACIÓN representa el código de la estación meteorológica que registró estos datos. La posición de este encabezado nos dice que el primer valor en cada línea será el código de la estación meteorológica. El encabezado NOMBRE indica que el segundo valor en cada línea es el nombre de la estación meteorológica que realizó la grabación. El resto de los encabezados especifican qué tipo de información se registró en cada lectura. Los datos que más nos interesan por ahora son la fecha, la temperatura alta (TMAX) y la temperatura baja (TMIN). Este es un conjunto de datos simple que contiene solo datos relacionados con la precipitación y la temperatura. Cuando descarga sus propios datos meteorológicos, puede optar por incluir una serie de otras medidas relacionadas con la velocidad del viento, la dirección y datos de precipitación más detallados.

Imprimir los encabezados y sus posiciones

Para facilitar la comprensión de los datos del encabezado del archivo, imprimimos cada encabezado y su posición en la lista:

sitka_highs.py

```
--recorte--
con abierto (nombre de archivo) como f:
    lector = csv.lector(f)
    header_row = siguiente (lector)

u para índice, column_header en enumerar (header_row):
        imprimir (índice, encabezado_columna)
```

La función enumerar () devuelve tanto el índice de cada elemento como el valor de cada elemento a medida que recorre una lista u. (Tenga en cuenta que hemos eliminado la línea print(header_row) a favor de esta versión más detallada).

Aquí está el resultado que muestra el índice de cada encabezado:

```
0 ESTACIÓN
1 NOMBRE
2 FECHA
3 PRCP
4 TAVG
5 TMAX
6 TMIN
```

Aquí vemos que las fechas y sus altas temperaturas se almacenan en las columnas 2 y 5. Para explorar estos datos, procesaremos cada fila de datos en *sitka_weather_07-2018_simple.csv* y extraeremos los valores con los índices 2 y 5.

Extracción y lectura de datos

Ahora que sabemos qué columnas de datos necesitamos, leamos algunos de esos datos. Primero, leeremos en la temperatura alta de cada día:

sitka_highs.py

```
--recorte--
con abierto (nombre de archivo) como f:
    lector = csv.lector(f)
    header_row = siguiente (lector)

    # Obtenga altas temperaturas de este archivo.
u   alto = []
v   para la fila en el lector:
w       alto = int(fila[5])
        highs.append (alto)

imprimir (altos)
```

Hacemos una lista vacía llamada highs u y luego recorremos las filas restantes en el archivo v. El objeto del lector continúa desde donde lo dejó en el archivo CSV y automáticamente regresa cada línea siguiendo su posición actual.
Debido a que ya hemos leído la fila del encabezado, el ciclo comenzará en la segunda línea donde comienzan los datos reales. En cada pasada por el bucle, extraemos los datos del índice 5, que corresponde al encabezado TMAX, y los asignamos a la variable high w. Usamos la función int() para convertir los datos, que se almacenan como una cadena, a un formato numérico para que podamos usarlos. Luego agregamos este valor a los máximos.

El siguiente listado muestra los datos ahora almacenados en máximos:

```
[62, 58, 70, 70, 67, 59, 58, 62, 66, 59, 56, 63, 65, 58, 56, 59, 64, 60, 60,
 61, 65, 65, 63, 59, 64, 65, 68, 66, 64, 67, 65]
```

Hemos extraído la temperatura alta para cada fecha y almacenado cada valor en una lista. Ahora vamos a crear una visualización de estos datos.

Trazado de datos en un gráfico de temperatura

Para visualizar los datos de temperatura que tenemos, primero crearemos un gráfico simple de los máximos diarios usando Matplotlib, como se muestra aquí:

sitka_highs.py

```
importar csv

importar matplotlib.pyplot como plt

nombre de archivo = 'datos/sitka_weather_07-2018_simple.csv'
con abierto (nombre de archivo) como f:
    --recorte--
```

```
    # Grafique las temperaturas altas.
    plt.style.use('nacido en el mar')
    higo, hacha = plt.subplots()
u ax.plot(highs, c='red')

    # Dar formato a la trama.
v plt.title("Temperaturas máximas diarias, julio de 2018", tamaño de fuente=24)
w plt.xlabel('', tamaño de fuente=16)
    plt.ylabel("Temperatura (F)", tamaño de fuente=16)
    plt.tick_params(axis='ambos', cual='principal', labelsize=16)

    plt.mostrar()
```

Pasamos la lista de máximos a plot() y pasamos c='red' para trazar los puntos en rojo u. (Graficaremos los máximos en rojo y los mínimos en azul.) Luego especificamos algunos otros detalles de formato, como el título, el tamaño de fuente y las etiquetas v, que debe reconocer del Capítulo 15. Porque todavía tenemos que agregue las fechas, no etiquetaremos el eje x, pero plt.xlabel() modifica el tamaño de fuente para hacer que las etiquetas predeterminadas sean más legibles w. La figura 16-1 muestra el gráfico resultante: un gráfico lineal simple de las temperaturas máximas de julio de 2018 en Sitka, Alaska.

Figura 16-1: Un gráfico de líneas que muestra las temperaturas máximas diarias de julio de 2018 en Sitka, Alaska

El módulo de fecha y hora

Agreguemos fechas a nuestro gráfico para que sea más útil. La primera fecha del archivo de datos meteorológicos se encuentra en la segunda fila del archivo:

```
"USW00025333","SITKA AIRPORT, AK US","2018-07-01","0.25","62","50"
```

Los datos se leerán como una cadena, por lo que necesitamos una forma de convertir la cadena "2018-07-01" en un objeto que represente esta fecha. Podemos construir un objeto que represente el 1 de julio de 2018 usando el método strptime() del módulo datetime . Veamos cómo funciona strptime() en una sesión de terminal:

```
>>> from datetime import datetime >>>
first_date = datetime.strptime('2018-07-01', '%Y-%m-%d')
>>> imprimir(primera_fecha)
2018-07-01 00:00:00
```

Primero importamos la clase de fecha y hora del módulo de fecha y hora. Luego llamamos al método strptime() usando la cadena que contiene la fecha con la que queremos trabajar como primer argumento. El segundo argumento le dice a Python cómo se formatea la fecha. En este ejemplo, Python interpreta que '%Y-' significa que la parte de la cadena antes del primer guión es un año de cuatro dígitos; '%m-' significa que la parte de la cadena antes del segundo guión es un número que representa el mes; y '%d' significa que la última parte de la cadena es el día del mes, del 1 al 31.

El método strptime() puede tomar una variedad de argumentos para determinar cómo interpretar la fecha. La tabla 16-1 muestra algunos de estos argumentos.

Tabla 16-1: Argumentos de formato de fecha y hora del módulo datetime

Argumento	Significado
%UN	Nombre del día de la semana, como lunes
%B	Nombre del mes, como enero
%metro	Mes, como un número (01 a 12)
%d	Día del mes, como un número (01 a 31)
%Y	Año de cuatro dígitos, como 2019
%y	Año de dos dígitos, como 19
%H	Hora, en formato de 24 horas (00 a 23)
%YO	Hora, en formato de 12 horas (01 a 12)
%pag	am o pm
%METRO	Minutos (00 a 59)
%S	Segundos (00 a 61)

Trazado de fechas

Ahora podemos mejorar nuestra gráfica de datos de temperatura extrayendo fechas para los máximos diarios y pasando esos máximos y fechas a plot(), como se muestra aquí:

sitka_highs.py

```
importar csv
desde fechahora fechahora de importación

importar matplotlib.pyplot como plt

nombre de archivo = 'datos/sitka_weather_07-2018_simple.csv'
```

```
con abierto (nombre de archivo)
        como f: lector = csv. lector
        (f) header_row = siguiente (lector)

        # Obtenga fechas y temperaturas altas de este archivo. u
fechas, máximos = [], [] para la fila en el lector: fecha_actual =
        fechahora.strptime(fila[2], '%Y-%m-%d') alto = int(fila[5])
en              fechas.append( fecha_actual ) highs.append(high)

    # Grafique las temperaturas altas.
    plt.style.use('seaborn') fig, ax =
    plt.subplots() w ax.plot(dates, highs,
c='red')

    # Dar formato a
    la trama. plt.title("Temperaturas máximas diarias, julio de 2018", tamaño de
    fuente=24) plt.xlabel('', tamaño de fuente=16) x fig.autofmt_xdate()

    plt.ylabel("Temperatura (F)", tamaño de fuente=16)
    plt.tick_params(eje='ambos', que='principal', tamaño de etiqueta=16)

    plt.mostrar()
```

Creamos dos listas vacías para almacenar las fechas y temperaturas altas del archivo u. Luego, convertimos los datos que contienen la información de fecha (fila [2]) en un objeto de fecha y hora v y lo agregamos a las fechas. Pasamos las fechas y los valores de temperatura máxima a plot() w. La llamada a fig.autofmt_xdate() x dibuja las etiquetas de fecha en diagonal para evitar que se superpongan. La figura 16-2 muestra el gráfico mejorado.

Figura 16-2: El gráfico es más significativo ahora que tiene fechas en el eje x.

Trazar un marco de tiempo más largo

Con la configuración de nuestro gráfico, agreguemos más datos para obtener una imagen más completa del clima en Sitka. Copie el archivo *sitka_weather_2018_simple.csv,* que contiene los datos meteorológicos de un año completo para Sitka, en la carpeta donde está almacenando los datos para los programas de este capítulo.

Ahora podemos generar un gráfico para el clima de todo el año:

sitka_highs.py

```
--recorte--
u nombre de archivo = 'datos/sitka_weather_2018_simple.csv'
con abierto (nombre de archivo) como f:
--recorte--
# Dar formato a la trama.
v plt.title("Temperaturas máximas diarias - 2018", tamaño de fuente=24)
plt.xlabel('', tamaño de fuente=16)
--recorte--
```

Modificamos el nombre del archivo para usar el nuevo archivo de datos *sitka_weather_2018_simple.csv* u, y actualizamos el título de nuestro gráfico para reflejar el cambio en su contenido v. La figura 16-3 muestra el gráfico resultante.

Figura 16-3: Datos de un año

Trazar una segunda serie de datos

Podemos hacer que nuestro gráfico informativo sea aún más útil al incluir las bajas temperaturas. Necesitamos extraer las bajas temperaturas del archivo de datos y luego agregarlas a nuestro gráfico, como se muestra aquí:

Sitka_highs_lows.py

```
--recorte--
nombre de archivo = 'sitka_weather_2018_simple.csv'
```

```
    con abierto (nombre de archivo)
        como f: lector = csv. lector (f)
        header_row = siguiente (lector)

        # Obtenga fechas y temperaturas altas y bajas de este archivo.
u fechas, máximos, mínimos = [], [], []
        para la fila en el lector:
            fecha_actual = fechahora.strptime(fila[2], '%Y-%m-%d') alta = int(fila[5])
            baja = int(fila[6]) fechas.append(fecha_actual ) highs.append (alto)
en          lows.append (bajo)

    # Grafique las temperaturas altas y bajas.
    plt.style.use('seaborn') fig, ax = plt.subplots()
    ax.plot(fechas, máximos, c='rojo') w ax.plot(fechas,
    mínimos, c='azul')

    # Dar formato a la
trama. x plt.title("Temperaturas máximas y mínimas diarias - 2018", tamaño de fuente=24)
    --recorte--
```

En u añadimos la lista vacía mínimos para mantener bajas temperaturas, y luego extrae y almacena la temperatura baja para cada fecha desde la séptima posición en cada fila (fila[6]) v. En w agregamos una llamada a plot() para las temperaturas bajas y coloreamos estos valores de azul. Finalmente, actualizamos el título x. La figura 16-4 muestra el gráfico resultante.

Figura 16-4: Dos series de datos en la misma parcela

Sombrear un área en el gráfico

Habiendo agregado dos series de datos, ahora podemos examinar el rango de temperaturas para cada día. Agreguemos un toque final al gráfico usando sombreado para mostrar el rango entre las temperaturas máximas y mínimas de cada día. Para hacerlo, usaremos el método fill_ between() , que toma una serie de valores x y dos series de valores y, y llena el espacio entre las dos series de valores y:

Sitka_highs_lows.py

```
--recorte--
# Grafique las temperaturas altas y bajas.
plt.style.use('nacido en el mar')
higo, hacha = plt.subplots()
u ax.plot(fechas, máximos, c='rojo', alfa=0.5)
ax.plot(fechas, mínimos, c='azul', alfa=0.5)
v plt.fill_ between(fechas, máximos, mínimos, color de cara='azul', alfa=0,1)
--recorte--
```

El argumento alfa en u controla la transparencia de un color. Un valor alfa de 0 es completamente transparente y 1 (el valor predeterminado) es completamente opaco. Al establecer alfa en 0,5, hacemos que las líneas rojas y azules de la trama parezcan más claras.

En v pasamos fill_ between() la lista de fechas para los valores x y luego las dos series altas y bajas de valores y. El argumento facecolor determina el color de la región sombreada; le damos un valor alfa bajo de 0,1 para que la región rellena conecte las dos series de datos sin distraer la atención de la información que representan. La figura 16-5 muestra el gráfico con la región sombreada entre los máximos y mínimos.

Figura 16-5: La región entre los dos conjuntos de datos está sombreada.

El sombreado ayuda a que el rango entre los dos conjuntos de datos sea inmediatamente evidente.

Comprobación de errores

Deberíamos poder ejecutar el código *sitka_highs_lows.py* usando datos para cualquier ubicación. Sin embargo, algunas estaciones meteorológicas recopilan datos diferentes a otras y, en ocasiones, funcionan mal y no recopilan algunos de los datos que se supone que deben recopilar. La falta de datos puede dar lugar a excepciones que bloqueen nuestros programas a menos que los gestionemos correctamente.

Por ejemplo, veamos qué sucede cuando intentamos generar un elemento diagrama de temperatura para Death Valley, California. Copie el archivo *death_valley_2018_simple.csv* a la carpeta donde está almacenando los datos para los programas de este capítulo.

Primero, ejecutemos el código para ver los encabezados que se incluyen en este archivo de datos:

Valle de la Muerte _altos_bajos.py

```
importar csv

nombre de archivo = 'datos/death_valley_2018_simple.csv'
con abierto (nombre de archivo) como f:
    lector = csv.lector(f)
    header_row = siguiente (lector)

    para el índice, column_header en enumerar (header_row):
        imprimir (índice, encabezado_columna)
```

Aquí está la salida:

```
0 ESTACIÓN
1 NOMBRE
2 FECHA
3 PRCP
4 TMAX
5 TMIN
6 TOBOS
```

La fecha está en la misma posición en el índice 2. Pero las temperaturas máxima y mínima están en los índices 4 y 5, por lo que tendríamos que cambiar los índices en nuestro código para reflejar estas nuevas posiciones. En lugar de incluir una lectura de temperatura promedio para el día, esta estación incluye TOBS, una lectura para un tiempo de observación específico.

Eliminé una de las lecturas de temperatura de este archivo para mostrar lo que sucede cuando faltan algunos datos de un archivo. Cambiar *sitka_highs_lows.py* para generar un gráfico para el Valle de la Muerte utilizando los índices que acabamos de señalar y ver qué sucede:

Valle de la Muerte _altos_bajos.py

```
--recorte--
nombre de archivo = 'datos/death_valley_2018_simple.csv'
con abierto (nombre de archivo) como f:
    --recorte--
    # Obtenga fechas y temperaturas altas y bajas de este archivo.
    fechas, máximos, mínimos = [], [], []
    para la fila en el lector:
        fecha_actual = fechahora.strptime(fila[2], '%Y-%m-%d')
```

```
en              alto = int(fila[4])
                bajo = int(fila[5])
                fechas.append(fecha_actual)
--recorte--
```

En u actualizamos los índices para que correspondan a los TMAX y TMIN de este archivo posiciones.

Cuando ejecutamos el programa, obtenemos un error, como se muestra en la última línea del siguiente resultado:

```
Rastreo (última llamada más reciente): Archivo
  "death_valley_highs_lows.py", línea 15, en <módulo> alto = int(fila[4])
                                                                          ''
ValueError: literal no válido para int() con base 10:
```

El rastreo nos dice que Python no puede procesar la temperatura alta de una de las fechas porque no puede convertir una cadena vacía ('') en un número entero. En lugar de revisar los datos y descubrir qué lectura falta, solo manejaremos los casos de datos faltantes directamente.

Ejecutaremos un código de verificación de errores cuando los valores se lean desde el Archivo CSV para manejar las excepciones que puedan surgir. Así es como funciona:

Valle de la Muerte
_altos_bajos.py

```
--recorte--
nombre de archivo = 'datos/death_valley_2018_simple.csv'
con abierto (nombre de archivo) como f:
    --recorte--
    para la fila en el lector:
        fecha_actual = fechahora.strptime(fila[2], '%Y-%m-%d')
en      tratar:
            alto = int(fila[4])
            bajo = int(fila[5])
        excepto ValueError:
            print(f"Datos faltantes para {fecha_actual}")
volvo   demás:
            fechas.append(fecha_actual)
            highs.append (alto)
            mínimos.append(bajo)

# Grafique las temperaturas altas y bajas.
--recorte--

# Dar formato a la trama.
x title = "Temperaturas máximas y mínimas diarias - 2018\nDeath Valley, CA"
plt.title(título, tamaño de fuente=20)
plt.xlabel('', tamaño de fuente=16)
--recorte--
```

Cada vez que examinamos una fila, tratamos de extraer la fecha y el alto y baja temperatura u. Si falta algún dato, Python generará un ValueError y lo manejamos imprimiendo un mensaje de error que incluye la fecha de los datos que faltan v. Después de imprimir el error, el ciclo continuará procesando la siguiente fila. Si todos los datos de una fecha se recuperan sin errores, el bloque else

se ejecutará y los datos se agregarán a las listas correspondientes w. Debido a que estamos trazando información para una nueva ubicación, actualizamos el título para incluir la ubicación en el gráfico y usamos un tamaño de fuente más pequeño para acomodar el título más largo x.

Cuando *ejecutes death_valley_highs_lows.py* ahora, verás que solo faltan datos en una fecha:

```
Datos faltantes para 2018-02-18 00:00:00
```

Debido a que el error se maneja adecuadamente, nuestro código puede generar un plot, que salta los datos que faltan. La figura 16-6 muestra el gráfico resultante.

Figura 16-6: Temperaturas máximas y mínimas diarias para el Valle de la Muerte

Comparando este gráfico con el gráfico de Sitka, podemos ver que Death Valley es más cálido en general que el sureste de Alaska, como esperamos. Además, la amplitud de temperaturas cada día es mayor en el desierto. La altura de la región sombreada deja esto claro.

Muchos conjuntos de datos con los que trabaja tendrán datos faltantes, mal formateados o incorrectos. Puede usar las herramientas que aprendió en la primera mitad de este libro para manejar estas situaciones. Aquí usamos un bloque try-except-else para manejar los datos faltantes. A veces usará continuar para omitir algunos datos o usar remove()
o del para eliminar algunos datos después de haberlos extraído. Utilice cualquier enfoque que funcione, siempre que el resultado sea una visualización significativa y precisa.

Descargar sus propios datos

Si desea descargar sus propios datos meteorológicos, siga estos pasos:

1. Visite el sitio de datos climáticos en línea de la NOAA en *https:// www.ncdc.noaa.gov / cdo-web/.* En la sección *Descubrir datos por* , haga clic en **Herramienta de búsqueda.** En el cuadro *Seleccionar un conjunto de datos* , elija **Resúmenes diarios.**

2. Seleccione un intervalo de fechas y, en la sección *Buscar ,* elija **Códigos postales.** Ingrese el código postal que le interesa y haga clic en **Buscar.**

3. En la página siguiente, verá un mapa y alguna información sobre el área te estás enfocando. Debajo del nombre de la ubicación, haga clic en **Ver detalles completos** o haga clic en el mapa y luego haga clic en **Detalles completos.**

4. Desplácese hacia abajo y haga clic en **Lista** de estaciones para ver las estaciones meteorológicas disponible en esta área. Elija una de las estaciones y haga clic **en Agregar al carrito.** Estos datos son gratuitos, aunque el sitio utiliza un ícono de carrito de compras. En el esquina superior derecha, haz clic en el carrito.

5. En *Seleccione la salida,* elija **Custom GHCN-Daily CSV.** Asegúrate que el intervalo de fechas es correcto y haga clic en **Continuar.**

6. En la página siguiente, puede seleccionar los tipos de datos que desea. Puede descargar un tipo de datos, por ejemplo, centrándose en la temperatura del aire, o puede descargar todos los datos disponibles de esta estación. Haz tu opciones y luego haga clic en **Continuar.**

7. En la última página, verá un resumen de su pedido. Entra tu dirección de correo electrónico y haga clic en **Enviar pedido.** Recibirás una confirmación que tu pedido fue recibido, y en unos minutos deberías recibir otro correo electrónico con un enlace para descargar sus datos.

Los datos que descargue se estructurarán igual que los datos con los que trabajamos con en esta sección. Puede tener encabezados diferentes a los que vio en esta sección. Pero si sigue los mismos pasos que usamos aquí, debería poder generar visualizaciones de los datos que le interesan.

Inténtalo tú mismo

16-1. Lluvia de Sitka: Sitka se encuentra en una selva tropical templada, por lo que recibe una buena cantidad de lluvia. En el archivo de datos sitka_weather_2018_simple.csv hay un encabezado llamado PRCP, que representa las cantidades de lluvia diarias. Realice una visualización centrándose en los datos de esta columna. Puede repetir el ejercicio para el Valle de la Muerte si siente curiosidad por la poca lluvia que cae en un desierto.

16-2. Comparación entre Sitka y el Valle de la Muerte: Las escalas de temperatura en los gráficos de Sitka y el Valle de la Muerte reflejan los diferentes rangos de datos. Para comparar con precisión el rango de temperatura en Sitka con el del Valle de la Muerte, necesita escalas idénticas en el eje y. Cambie la configuración del eje y en uno o ambos gráficos en las Figuras 16-5 y 16-6. Luego haga una comparación directa entre los rangos de temperatura en Sitka y el Valle de la Muerte (o dos lugares cualquiera que desee comparar).

16-3. San Francisco: ¿Son las temperaturas en San Francisco más como las temperaturas en Sitka o las temperaturas en el Valle de la Muerte? Descargue algunos datos de San Francisco y genere un gráfico de temperatura alta-baja para San Francisco para hacer una comparación.

16-4. Índices automáticos: en esta sección codificamos los índices correspondientes a las columnas TMIN y TMAX . Use la fila de encabezado para determinar los índices de estos valores, de modo que su programa pueda funcionar para Sitka o Death Valley. Use el nombre de la estación para generar automáticamente un título apropiado para su gráfico también.

16-5. Explorar: genere algunas visualizaciones más que examinen cualquier otro aspecto meteorológico que le interese para cualquier lugar que le interese.

Mapeo de conjuntos de datos globales: formato JSON

En esta sección, descargará un conjunto de datos que representan todos los terremotos que han ocurrido en el mundo durante el mes anterior. Luego, harás un mapa que muestre la ubicación de estos terremotos y la importancia de cada uno. Debido a que los datos se almacenan en formato JSON, trabajaremos con ellos usando el módulo json . Usando la herramienta de mapeo fácil de usar para principiantes de Plotly para datos basados en la ubicación, creará visualizaciones que muestran claramente la distribución global de los terremotos.

Descarga de datos de terremotos

Copie el archivo *eq_1_day_m1.json* en la carpeta donde está almacenando los datos de los programas de este capítulo. Los terremotos se clasifican por su magnitud en la escala de Richter. Este archivo incluye datos de todos los terremotos con una magnitud M1 o mayor que tuvieron lugar en las últimas 24 horas (en el momento de escribir este artículo). Estos datos provienen de una de las fuentes de datos de terremotos del Servicio Geológico de los Estados Unidos, que puede encontrar en *https:// earthquake.usgs .gov/ terremotos/ alimentación/.*

Examen de datos JSON

Cuando abra *eq_1_day_m1.json,* verá que es muy denso y difícil de leer:

```
{"tipo":"FeatureCollection","metadatos":{"generado":1550361461000,...
{"type":"Feature","properties":{"mag":1.2,"place":"11km NNE of Nor...
{"type":"Feature","properties":{"mag":4.3,"place":"69 km al NNO de Ayn...
{"type":"Feature","properties":{"mag":3.6,"place":"126km SSE of Co...
{"type":"Feature","properties":{"mag":2.1,"place":"21 km al NNO de Teh...
{"type":"Feature","properties":{"mag":4,"place":"57 km al suroeste de Kakto...
--recorte--
```

Este archivo está formateado más para máquinas que para humanos. Pero podemos ver que el archivo contiene algunos diccionarios, así como información que nos interesa, como la magnitud y la ubicación de los terremotos.

El módulo json proporciona una variedad de herramientas para explorar y trabajar con datos JSON. Algunas de estas herramientas nos ayudarán a reformatear el archivo para que podamos ver los datos sin procesar más fácilmente antes de comenzar a trabajar con ellos mediante programación.

Comencemos cargando los datos y mostrándolos en un formato que sea más fácil leer. Este es un archivo de datos largo, por lo que en lugar de imprimirlo, reescribiremos los datos en un archivo nuevo. Luego podemos abrir ese archivo y desplazarnos hacia adelante y hacia atrás fácilmente a través de los datos:

eq_explorar_datos.py

```
importar json

# Explore la estructura de los datos.
nombre de archivo = 'datos/eq_data_1_day_m1.json'
con abierto (nombre de archivo) como f:
u all_eq_data = json.load(f)

v readable_file = 'datos/readable_eq_data.json'
con abierto (archivo_legible, 'w') como f:
w json.dump(all_eq_data, f, sangría=4)
```

Primero importamos el módulo json para cargar los datos correctamente desde el archivo y luego almacenamos todo el conjunto de datos en all_eq_data u. El json.load() La función convierte los datos a un formato con el que Python puede trabajar: en este caso, un diccionario gigante. En v creamos un archivo para escribir estos mismos datos en un formato más legible. La función json.dump() toma un objeto de datos JSON y un objeto de archivo, y escribe los datos en ese archivo w. El argumento indent=4 le dice a dump() que formatee los datos usando una sangría que coincida con la estructura de los datos.

Cuando busca en su directorio de *datos* y abre el archivo *readable_eq_data .json,* aquí está la primera parte de lo que verá:

legible_eq_datos.json

```
{
    "tipo": "Colección de funciones",
en "metadatos": {
        "generado": 1550361461000,
        "url": "https://earthquake.usgs.gov/earthquakes/.../1.0_day.geojson",
        "title": "USGS Magnitud 1.0+ Terremotos, día pasado",
        "estado": 200,
        "api": "1.7.0",
        "contar": 158
    },
v "características": [
    --recorte--
```

La primera parte del archivo incluye una sección con la clave "metadatos". Este nos dice cuándo se generó el archivo de datos y dónde podemos encontrar los datos en línea. También nos da un título legible por humanos y el número de terremotos incluidos en este archivo. En este período de 24 horas se registraron 158 sismos.

Este archivo *geoJSON* tiene una estructura que resulta útil para los datos basados en la ubicación. La información se almacena en una lista asociada a las "características" clave v. Debido a que este archivo contiene datos de terremotos, los datos están en forma de lista donde cada elemento de la lista corresponde a un solo terremoto. Esta estructura puede parecer confusa, pero es bastante poderosa. Permite a los geólogos almacenar toda la información que necesitan en un diccionario sobre cada terremoto y luego juntar todos esos diccionarios en una gran lista.

Veamos un diccionario que representa un solo terremoto:

legible_eq_datos.json

```
--recorte--
    {
        "tipo": "Característica",
en      "propiedades": {
            "mag": 0.96,
            --recorte--
en          "title": "M 1.0 - 8 km al NE de Aguanga, CA"
        },
en      "geometría": {
            "tipo": "Punto",
            "coordenadas": [
xy              -116.7941667,
                33.4863333,
                3.22
            ]
        },
        "id": "ci37532978"
    },
```

Las "propiedades" clave contienen mucha información sobre cada terremoto u. Estamos principalmente interesados en la magnitud de cada terremoto, que está asociado con la clave "mag". También nos interesa el título de cada terremoto, que proporciona un buen resumen de su magnitud y ubicación v.

La "geometría" clave nos ayuda a entender dónde ocurrió el terremoto w. Necesitaremos esta información para mapear cada evento. Podemos encontrar la longitud x y la latitud y para cada terremoto en una lista asociada a las "coordenadas" clave.

Este archivo contiene mucho más anidamiento de lo que usaríamos en el código que escribimos, así que si parece confuso, no se preocupe: Python manejará la mayor parte de la complejidad. Solo trabajaremos con uno o dos niveles de anidamiento a la vez. Comenzaremos sacando un diccionario para cada terremoto que se registró en el período de 24 horas.

Cuando hablamos de ubicaciones, a menudo decimos primero la latitud de la ubicación, seguida de la longitud. Esta convención probablemente surgió porque los humanos descubrieron la latitud mucho antes de que desarrolláramos el concepto de longitud. Sin embargo, muchos marcos geoespaciales enumeran primero la longitud y luego la latitud, porque esto corresponde a la (x, y) *convención que usamos en las representaciones matemáticas. El formato geoJSON sigue la convención (longitud, latitud), y si usa un marco diferente, es importante saber qué convención sigue ese marco.*

Hacer una lista de todos los terremotos

Primero, haremos una lista que contenga toda la información sobre cada terremoto que ocurrió.

eq_explorar_datos.py

```
importar json

# Explore la estructura de los datos.
nombre de archivo = 'datos/eq_data_1_day_m1.json'
con abierto (nombre de archivo) como f:
    all_eq_data = json.load(f)

all_eq_dicts = all_eq_data['características']
imprimir (len (todos_eq_dicts))
```

Tomamos los datos asociados con las 'características' clave y los almacenamos en all_eq_dicts. Sabemos que este archivo contiene registros sobre 158 terremotos, y el resultado verifica que hemos capturado todos los terremotos en el archivo:

```
158
```

Observe lo corto que es este código. El archivo perfectamente formateado *readable_eq_data.json* tiene más de 6000 líneas. Pero en solo unas pocas líneas, podemos leer todos esos datos y almacenarlos en una lista de Python. A continuación, extraeremos las magnitudes de cada terremoto.

Extracción de magnitudes

Usando la lista que contiene datos sobre cada terremoto, podemos recorrer esa lista y extraer cualquier información que queramos. Ahora sacaremos la magnitud de cada terremoto:

eq_explorar_datos.py

```
--recorte--
all_eq_dicts = all_eq_data['características']

tu revistas = []
para eq_dict en all_eq_dicts:
v mag = eq_dict['propiedades']['mag']
    revistas.append(mag)

imprimir (revistas [: 10])
```

Hacemos una lista vacía para almacenar las magnitudes, y luego recorremos el diccionario all_eq_dicts u. Dentro de este ciclo, cada terremoto está representado por el diccionario eq_dict. La magnitud de cada terremoto se almacena en la sección 'propiedades' de este diccionario bajo la clave 'mag' v. Almacenamos cada magnitud en la variable mag y luego la agregamos a la lista mags.

Imprimimos las primeras 10 magnitudes, para que podamos ver si estamos obteniendo los datos correctos:

```
[0,96, 1,2, 4,3, 3,6, 2,1, 4, 1,06, 2,3, 4,9, 1,8]
```

A continuación, extraeremos los datos de ubicación de cada terremoto y luego podremos hacer un mapa de los terremotos.

Extracción de datos de ubicación

Los datos de ubicación se almacenan bajo la clave "geometría". Dentro del diccionario de geometría hay una clave de "coordenadas" , y los primeros dos valores en esta lista son la longitud y la latitud. Así es como extraeremos estos datos:

eq_explorar_datos.py

```
--recorte--
all_eq_dicts = all_eq_data['características']

Mags, lons, lats = [], [], []
para eq_dict en all_eq_dicts:
        mag = eq_dict['propiedades']['mag']
u lon = eq_dict['geometría']['coordenadas'][0]
        lat = eq_dict['geometría']['coordenadas'][1]
        revistas.append(mag)
        lons.append(lon)
        lats.append(lat)

imprimir (revistas [: 10])
imprimir(longitud[:5])
imprimir (lats [: 5])
```

Hacemos listas vacías para las longitudes y latitudes. El código eq_dict ['geometría'] accede al diccionario que representa el elemento geométrico del terremoto u. La segunda clave, 'coordenadas', extrae la lista de valores asociados con 'coordenadas'. Finalmente, el índice 0 solicita el primer valor en la lista de coordenadas, que corresponde a la longitud de un terremoto.

Cuando imprimimos las primeras cinco longitudes y latitudes, la salida muestra que estamos extrayendo los datos correctos:

```
[0,96, 1,2, 4,3, 3,6, 2,1, 4, 1,06, 2,3, 4,9, 1,8]
[-116.7941667, -148.9865, -74.2343, -161.6801, -118.5316667]
[33.4863333, 64.6673, -12.1025, 54.2232, 35.3098333]
```

Con estos datos, podemos pasar a mapear cada terremoto.

Construyendo un mapa del mundo

Con la información que hemos obtenido hasta ahora, podemos construir un mapa del mundo simple. Aunque todavía no se verá presentable, queremos asegurarnos de que la información se muestre correctamente antes de centrarnos en cuestiones de estilo y presentación.

Aquí está el mapa inicial:

eq_world_map.py

```
importar json

u de plotly.graph_objs import Scattergeo, Layout
desde plotly importar sin conexión

--recorte--
```

```
    para eq_dict en all_eq_dicts:
        --recorte--

    # Mapa de los terremotos.
v data = [Scattergeo(lon=lons, lat=lats)]
w my_layout = Layout(title='Terremotos Globales')

x fig = {'datos': datos, 'diseño': mi_diseño}
    offline.plot(fig, filename='global_earthquakes.html')
```

Importamos el tipo de gráfico Scattergeo y la clase Layout , y luego importamos el módulo fuera de línea para representar el mapa u. Como hicimos al hacer un gráfico de barras, definimos una lista llamada datos. Creamos el objeto Scattergeo dentro de esta lista v, porque puede trazar más de un conjunto de datos en cualquier visualización que realice. Un tipo de gráfico Scattergeo le permite superponer un gráfico de dispersión de datos geográficos en un mapa. En el uso más simple de este tipo de gráfico, solo necesita proporcionar una lista de longitudes y una lista de latitudes.

Le damos al gráfico un título apropiado w y creamos un diccionario llamado fig que contiene los datos y el diseño x. Por último, pasamos fig a la función plot() junto con un nombre de archivo descriptivo para la salida. Cuando ejecute este archivo, debería ver un mapa similar al de la Figura 16-7. Los terremotos generalmente ocurren cerca de los límites de las placas, lo que coincide con lo que vemos en el gráfico.

Figura 16-7: Un mapa simple que muestra dónde ocurrieron todos los terremotos en las últimas 24 horas

Podemos hacer muchas modificaciones para que este mapa sea más significativo y fácil de leer, así que hagamos algunos de estos cambios.

Una forma diferente de especificar los datos del gráfico

Antes de configurar el gráfico, veamos una forma ligeramente diferente de especificar los datos para un gráfico de Plotly. En el gráfico actual, la lista de datos se define en una línea:

```
data = [Scattergeo(lon=lons, lat=lats)]
```

Esta es una de las formas más sencillas de definir los datos de un gráfico en Plotly. Pero no es la mejor manera cuando desea personalizar la presentación. Aquí hay una forma equivalente de definir los datos para el gráfico actual:

```
datos = [{
    'tipo': 'disperso',
    'lon': lons,
    'lat': lats,
}]
```

En este enfoque, toda la información sobre los datos se estructura como pares clave-valor en un diccionario. Si coloca este código en *eq_plot.py,* verá el mismo gráfico que acabamos de generar. Este formato nos permite especificar personalizaciones más fácilmente que el formato anterior.

Personalización del tamaño del marcador

Cuando buscamos cómo mejorar el estilo del mapa, debemos centrarnos en los aspectos de los datos que queremos comunicar más claramente. El mapa actual muestra la ubicación de cada terremoto, pero no comunica la gravedad de ningún terremoto. Queremos que los espectadores vean de inmediato dónde ocurren los terremotos más importantes del mundo.

Para ello, cambiaremos el tamaño de los marcadores en función de la magni tud de cada terremoto:

eq_world_map.py

```
importar json
--recorte--
# Mapa de los terremotos.
datos = [{
    'tipo': 'disperso',
    'lon': lons,
    'lat': lats,
u 'marcador': {
en          'tamaño': [5 * revista para revista en revistas],
    },
}]
my_layout = Layout(title='Terremotos Globales')
--recorte--
```

Plotly ofrece una gran variedad de personalizaciones que puede realizar en una serie de datos, cada una de las cuales se puede expresar como un par clave-valor. Aquí estamos usando la tecla 'marcador' para especificar qué tan grande debe ser cada marcador en el mapa. Usamos un diccionario anidado como el valor asociado con 'marcador', porque puede especificar una serie de configuraciones para todos los marcadores en una serie.

Queremos que el tamaño corresponda a la magnitud de cada terremoto. Pero si solo pasamos la lista de revistas , los marcadores serían demasiado pequeños para ver fácilmente las diferencias de tamaño. Necesitamos multiplicar la magnitud por un factor de escala para obtener un tamaño de marcador apropiado. En mi pantalla, un valor de 5 funciona bien; un valor un poco más pequeño o más grande podría funcionar mejor para su mapa. Usamos una lista de comprensión, que genera un tamaño de marcador apropiado para cada valor en la lista de revistas v.

Cuando ejecute este código, debería ver un mapa que se parece al en la figura 16-8. Este es un mapa mucho mejor, pero aún podemos hacer más.

Figura 16-8: El mapa ahora muestra la magnitud de cada terremoto.

Personalización de colores de marcador

También podemos personalizar el color de cada marcador para proporcionar una clasificación de la gravedad de cada terremoto. Usaremos las escalas de color de Plotly para hacer esto. Antes de realizar estos cambios, copie el archivo *eq_data_30_day_m1.json* a su directorio de datos. Este archivo incluye datos de terremotos durante un período de 30 días, y el mapa será mucho más interesante de ver usando este conjunto de datos más grande.

Aquí se explica cómo usar una escala de colores para representar la magnitud de cada terremoto:

eq_world_map.py

```
    --recorte--
u nombre de archivo = 'datos/eq_data_30_day_m1.json'
    --recorte--
    # Mapa de los terremotos.
    datos = [{
        --recorte--
```

```
        'marcador': {
            'tamaño': [5 * revista para revista en revistas],
            'color': revistas,
            'escala de colores': 'Viridis',
            'escala inversa': Cierto,
vwxy        'barra de colores': {'título': 'Magnitud'},
        },
    }]
    --recorte--
```

Asegúrese de actualizar el nombre del archivo para usar el conjunto de datos de 30 días u. Todos los cambios significativos aquí ocurren en el diccionario de 'marcadores' , porque solo estamos modificando la apariencia de los marcadores. La configuración de 'color' le dice a Plotly qué valores debe usar para determinar dónde cae cada marcador en la escala de colores v. Usamos la lista de revistas para determinar el color que se usa.

La configuración de 'escala de colores ' le dice a Plotly qué rango de colores usar: 'Viridis' es una escala de colores que va del azul oscuro al amarillo brillante y funciona bien para este conjunto de datos w. Establecemos 'reversescale' en True, porque queremos usar amarillo brillante para los valores más bajos y azul oscuro para los terremotos x más severos. La configuración de la 'barra de colores' nos permite controlar la apariencia de la escala de colores que se muestra en el lateral del mapa. Aquí llamamos a la escala de colores 'Magnitud' para dejar claro qué representan los colores x.

Cuando ejecute el programa ahora, verá un mapa mucho más atractivo. En la Figura 16-9, la escala de colores muestra la severidad de los terremotos individuales. ¡Trazar tantos terremotos realmente deja en claro dónde están los límites de las placas tectónicas!

Figura 16-9: En 30 días de terremotos, el color y el tamaño se utilizan para representar la magnitud de cada terremoto.

Otras escalas de colores

También puede elegir entre una serie de otras escalas de colores. Para ver las escalas de colores disponibles, guarde el siguiente programa corto como *show_color_scales.py:*

mostrar_color _balanzas.py

```
de plotly importar colores

para clave en colors.PLOTLY_SCALES.keys():
    imprimir (clave)
```

Plotly almacena las escalas de colores en el módulo de colores. Las escalas de colores se definen en el diccionario PLOTLY_SCALES, y los nombres de las escalas de colores sirven como claves en el diccionario. Aquí está el resultado que muestra todas las escalas de color disponibles:

```
grises
YlGnBu
Verduras
--recorte--
Viridis
```

Siéntase libre de probar estas escalas de colores; recuerda que puedes revertir cualquiera de estas escalas usando la configuración de escala inversa .

Si imprime el diccionario *PLOTLY_SCALES , puede ver cómo se definen las escalas de color. Cada escala tiene un color inicial y un color final, y algunas escalas también tienen uno o más colores intermedios definidos. Plotly interpola sombras entre cada uno de estos colores definidos.*

Agregar texto flotante

Para finalizar este mapa, agregaremos un texto informativo que aparece cuando pasas el cursor sobre el marcador que representa un terremoto. Además de mostrar la longitud y la latitud, que aparecen de forma predeterminada, mostraremos la magnitud y también proporcionaremos una descripción de la ubicación aproximada.

Para realizar este cambio, debemos extraer un poco más de datos del archivo y agregarlos también al diccionario en datos :

eq_world_map.py

```
--recorte--
umags, lons, lats, hover_texts = [], [], [], []
para eq_dict en all_eq_dicts:
    --recorte--
    lat = eq_dict['geometría']['coordenadas'][1]
v título = eq_dict['propiedades']['título']
    revistas.append(mag)
    lons.append(lon)
    lats.append(lat)
    hover_texts.append(título)
--recorte--
```

```
# Mapa de los terremotos.
datos = [{
        'tipo': 'disperso',
        'lon': lons,
        'lat': lats,
w 'texto': hover_texts,
        'marcador': {
                --recorte--
        },
}]
--recorte--
```

Primero creamos una lista llamada hover_texts para almacenar la etiqueta que usaremos para cada marcador u. La sección de "título" de los datos del terremoto contiene un nombre descriptivo de la magnitud y ubicación de cada terremoto además de su longitud y latitud. En v extraemos esta información y la asignamos al título de la variable, y luego la agregamos a la lista hover_texts.

Cuando incluimos la clave 'texto' en el objeto de datos , Plotly usa este valor como una etiqueta para cada marcador cuando el espectador se desplaza sobre el marcador. Cuando pasamos una lista que coincide con el número de marcadores, Plotly extrae una etiqueta individual para cada marcador que genera w. Cuando ejecuta este programa, debería poder pasar el cursor sobre cualquier marcador, ver una descripción de dónde ocurrió ese terremoto y leer su magnitud exacta.

¡Esto es impresionante! En aproximadamente 40 líneas de código, hemos creado un mapa visualmente atractivo y significativo de la actividad sísmica global que también ilustra la estructura geológica del planeta. Plotly ofrece una amplia gama de formas en las que puede personalizar la apariencia y el comportamiento de sus visualizaciones. Con las numerosas opciones de Plotly, puede crear gráficos y mapas que muestren exactamente lo que usted desea.

Inténtalo tú mismo

16-6. Refactorización: el ciclo que extrae datos de all_eq_dicts usa variables para la magnitud, longitud, latitud y título de cada terremoto antes de agregar estos valores a sus listas apropiadas. Se eligió este enfoque para aclarar cómo extraer datos de un archivo JSON, pero no es necesario en su código.
En lugar de usar estas variables temporales, extraiga cada valor de eq_dict y agréguelo a la lista correspondiente en una línea. Hacerlo debería acortar el cuerpo de este ciclo a solo cuatro líneas.

16-7. Título automatizado: en esta sección, especificamos el título manualmente al definir my_layout, lo que significa que debemos recordar actualizar el título cada vez que cambia el archivo de origen. En su lugar, puede usar el título del conjunto de datos en la parte de metadatos del archivo JSON. Obtenga este valor, asígnelo a una variable y utilícelo para el título del mapa cuando esté definiendo my_layout.

(continuado)

16-8. Terremotos recientes: puede encontrar archivos de datos que contienen información sobre los terremotos más recientes en períodos de 1 hora, 1 día, 7 días y 30 días en línea. Vaya a https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php y verá una lista de enlaces a conjuntos de datos para varios períodos de tiempo, centrándose en terremotos de diferentes magnitudes. Descargue uno de estos conjuntos de datos y cree una visualización de la actividad sísmica más reciente.

16-9. World Fires: En los recursos de este capítulo, encontrará un archivo llamado world_fires_1_day.csv. Este archivo contiene información sobre los incendios que arden en diferentes lugares del mundo, incluidas la latitud y la longitud, y el brillo de cada incendio. Usando el trabajo de procesamiento de datos de la primera parte de este capítulo y el trabajo de mapeo de esta sección, haz un mapa que muestre qué partes del mundo están afectadas por incendios.

Puede descargar versiones más recientes de estos datos en https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/active-fire-data/. Puede encontrar enlaces a los datos en formato CSV en la sección TXT.

Resumen

En este capítulo, aprendió a trabajar con conjuntos de datos del mundo real. Procesó archivos CSV y JSON, y extrajo los datos en los que desea concentrarse. Usando datos meteorológicos históricos, aprendió más sobre cómo trabajar con Matplotlib, incluido cómo usar el módulo de fecha y hora y cómo trazar varias series de datos en un gráfico. Trazó datos geográficos en un mapa mundial en Plotly y también diseñó mapas y gráficos de Plotly.

A medida que adquiera experiencia trabajando con archivos CSV y JSON, podrá para procesar casi cualquier dato que desee analizar. Puede descargar la mayoría de los conjuntos de datos en línea en cualquiera de estos formatos o en ambos. Al trabajar con estos formatos, también podrá aprender a trabajar con otros formatos de datos más fácilmente.

En el próximo capítulo, escribirá programas que recopilan automáticamente sus propios datos de fuentes en línea y luego creará visualizaciones de esos datos. Estas son habilidades divertidas si quieres programar como pasatiempo y habilidades críticas si estás interesado en programar profesionalmente.

17

En este capítulo, aprenderá a escribir un programa autónomo que genere un visualización basada en los datos que recupera. Su El programa utilizará una *interfaz de programación de aplicaciones web (API)* para solicitar automáticamente información específica de un sitio web, en lugar de páginas enteras, y luego usará esa información para generar una visualización. Debido a que los programas escritos de esta manera siempre usarán datos actuales para generar una visualización, incluso cuando esos datos puedan cambiar rápidamente, siempre estarán actualizados.

Una API web es una parte de un sitio web diseñado para interactuar con programas. Esos programas utilizan direcciones URL muy específicas para solicitar cierta información. Este tipo de solicitud se denomina *llamada API.* Los datos solicitados serán devueltos en un

Uso de una API web   Trabajar con API

formato fácil de procesar, como JSON o CSV. La mayoría de las aplicaciones que se basan en fuentes de datos externas, como las aplicaciones que se integran con sitios de redes sociales, se basan en llamadas API.

Git y GitHub

Basaremos nuestra visualización en información de GitHub, un sitio que permite a los programadores colaborar en proyectos de codificación. Usaremos la API de GitHub para solicitar información sobre los proyectos de Python en el sitio y luego generaremos una visualización interactiva de la popularidad relativa de estos proyectos usando Plotly.

GitHub *(https:// github.com/)* toma su nombre de Git, un sistema de control de versiones distribuido. Git ayuda a las personas a administrar su trabajo en un proyecto, por lo que los cambios realizados por una persona no interferirán con los cambios que realicen otras personas. Cuando implementa una nueva función en un proyecto, Git realiza un seguimiento de los cambios que realiza en cada archivo. Cuando su nuevo código funciona, *confirma* los cambios que ha realizado y Git registra el nuevo estado de su proyecto. Si comete un error y desea revertir los cambios, puede volver fácilmente a cualquier estado de funcionamiento anterior. (Para obtener más información sobre el control de versiones con Git, consulte el Apéndice D). Los proyectos en GitHub se almacenan en *repositorios* que contienen todo lo relacionado con el proyecto: su código, información sobre sus colaboradores, cualquier problema o informe de error, etc.

Cuando a los usuarios de GitHub les gusta un proyecto, pueden "destacarlo" para mostrar su apoyo y realizar un seguimiento de los proyectos que podrían querer usar. En este capítulo, escribiremos un programa para descargar automáticamente información sobre los proyectos de Python con más estrellas en GitHub y luego crearemos una visualización informativa de estos proyectos.

Solicitud de datos mediante una llamada a

la API La API de GitHub le permite solicitar una amplia gama de información a través de llamadas a la API. Para ver cómo se ve una llamada API, ingrese lo siguiente en la barra de direcciones de su navegador y presione enter:

https://api.github.com/search/repositories?q=language:python&sort=stars

Esta llamada devuelve la cantidad de proyectos de Python alojados actualmente en GitHub, así como información sobre los repositorios de Python más populares. Examinemos la llamada. La primera parte, https://api.github.com/, dirige la solicitud a la parte de GitHub que responde a las llamadas a la API. La siguiente parte, búsqueda/repositorios, le dice a la API que realice una búsqueda a través de todos los repositorios en GitHub.

El signo de interrogación después de los repositorios indica que estamos a punto de pasar una discusión. La q significa *consulta* y el signo igual (=) nos permite comenzar a especificar una consulta (q=). Al usar language:python, indicamos que queremos información solo sobre los repositorios que tienen Python como idioma principal. La parte final, &sort=stars, ordena los proyectos por el número de estrellas que se les han otorgado.

El siguiente fragmento muestra las primeras líneas de la respuesta.

```
{
u "recuento_total": 3494012,
v "resultados_incompletos": falso,
w "elementos": [
        {
            "identificación": 21289110,
            "id_nodo": "MDEwOlJlcG9zaXRvcnkyMTI4OTExMA==",
            "nombre": "pitón impresionante",
            "full_name": "vinta/genial-python",
            --recorte--
```

Puede ver en la respuesta que esta URL no está destinada principalmente ser ingresado por humanos, porque está en un formato destinado a ser procesado por un programa. GitHub encontró 3,494,012 proyectos de Python a partir de este escrito. Debido a que el valor de "incomplete_results" es falso, sabemos que la solicitud fue exitosa (no está incompleta) v. Si GitHub no hubiera podido procesar completamente la solicitud de la API, habría regresado verdadero aquí. Los "elementos" devueltos se muestran en la lista a continuación, que contiene detalles sobre los proyectos de Python más populares en GitHub w.

Solicitudes de instalación

El paquete Requests permite que un programa de Python solicite fácilmente información de un sitio web y examine la respuesta. Use pip para instalar Solicitudes:

```
$ python -m pip install --solicitudes de usuario
```

Esta línea le dice a Python que ejecute el módulo pip e instale el paquete Requests en la instalación de Python del usuario actual. Si usa **python3** o un comando diferente cuando ejecuta programas o instala paquetes, asegúrese de usar el mismo comando aquí.

Si este comando no funciona en macOS, intente ejecutar el comando nuevamente sin la marca --user .

Procesamiento de una respuesta de API

Ahora comenzaremos a escribir un programa para emitir automáticamente una llamada a la API y procesar los resultados mediante la identificación de los proyectos de Python con más estrellas en GitHub:

python_repos.py

```
u solicitudes de importación

    # Realice una llamada a la API y almacene la respuesta.
v url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'
w encabezados = {'Aceptar': 'aplicación/vnd.github.v3+json'}
xr = solicitudes.get(url, encabezados=encabezados)
y print(f"Código de estado: {r.status_code}")
```

```
    # Almacena la respuesta de la API en una variable.
z respuesta_dict = r.json()

    # Resultados del proceso.
    imprimir (response_dict.keys())
```

En u importamos el módulo de solicitudes . En v almacenamos la URL de la llamada API en la variable url . GitHub se encuentra actualmente en la tercera versión de su API, por lo que definimos encabezados para la llamada a la API que solicitan explícitamente usar esta versión de la API. Luego usamos solicitudes para hacer la llamada a la API x.

Llamamos a get() y le pasamos la URL y el encabezado que definimos, y asignamos el objeto de respuesta a la variable r. El objeto de respuesta tiene un atributo llamado status_code, que nos dice si la solicitud fue exitosa. (Un código de estado de 200 indica una respuesta exitosa). En y, imprimimos el valor de status_code para asegurarnos de que la llamada se realizó correctamente.

La API devuelve la información en formato JSON, por lo que usamos el json() método para convertir la información a un diccionario de Python z. Almacenamos el diccionario resultante en response_dict.

Finalmente, imprimimos las claves de response_dict y vemos este resultado:

```
Código de estado: 200
dict_keys(['total_count', 'incomplete_results', 'items'])
```

Como el código de estado es 200, sabemos que la solicitud fue exitosa.
El diccionario de respuestas contiene solo tres claves: 'total_count', 'incomplete _resultados' y 'elementos'. Echemos un vistazo dentro del diccionario de respuestas.

Las llamadas simples como esta deberían devolver un conjunto completo de resultados, por lo que es seguro ignorar el valor asociado con 'incomplete_results'. Pero cuando realiza llamadas API más complejas, su programa debe verificar este valor.

Trabajar con el diccionario de respuestas

Con la información de la llamada API almacenada como diccionario, podemos trabajar con los datos almacenados allí. Generemos una salida que resuma la información. Esta es una buena manera de asegurarnos de que recibimos la información que esperábamos y de comenzar a examinar la información que nos interesa:

python_repos.py

```
    solicitudes de importación

    # Realice una llamada a la API y almacene la respuesta.
    --recorte--

    # Almacena la respuesta de la API en una variable.
    respuesta_dict = r.json()
u print(f"Repositorios totales: {response_dict['total_count']}")

    # Explorar información sobre los repositorios.
v repo_dicts = response_dict['elementos']
    print(f"Repositorios devueltos: {len(repo_dicts)}")
```

```
    # Examine el primer repositorio.
w repo_dict = repo_dicts[0]
x print(f"\nTeclas: {len(repo_dict)}")
y para la clave ordenada (repo_dict.keys()):
        imprimir (clave)
```

En u imprimimos el valor asociado con 'total_count', que representa el número total de repositorios de Python en GitHub.

El valor asociado con 'elementos' es una lista que contiene varios diccionarios, cada uno de los cuales contiene datos sobre un repositorio individual de Python. En v almacenamos esta lista de diccionarios en repo_dicts. Luego imprimimos la longitud de repo_dicts para ver de cuántos repositorios tenemos información.

Para ver más de cerca la información devuelta sobre cada repositorio, extraemos el primer elemento de repo_dicts y lo almacenamos en repo_dict w. Luego imprimimos el número de claves en el diccionario para ver cuánta información tenemos x. En y imprimimos todas las claves del diccionario para ver qué tipo de información
ción está incluida.

Los resultados nos dan una imagen más clara de los datos reales:

```
    Código de estado: 200
    Repositorios totales: 3494030
    Repositorios devueltos: 30

u Teclas: 73
    archivo_url
    archivado
    asignados_url
    --recorte--
    URL
    vigilantes
    contador_de_observadores
```

La API de GitHub devuelve mucha información sobre cada repositorio: hay 73 claves en repo_dict u. Cuando revise estas claves, tendrá una idea del tipo de información que puede extraer sobre un proyecto. (La única forma de saber qué información está disponible a través de una API es leer la documentación o examinar la información a través del código, como lo estamos haciendo aquí).

Extraigamos los valores de algunas de las claves en repo_dict:

python_repos.py

```
    --recorte--
    # Explorar información sobre los repositorios.
    repo_dicts = respuesta_dict['elementos']
    print(f"Repositorios devueltos: {len(repo_dicts)}")

    # Examine el primer repositorio.
    repo_dict = repo_dicts[0]

    print("\nInformación seleccionada sobre el primer repositorio:")
u print(f"Nombre: {repo_dict['nombre']}")
v print(f"Propietario: {repo_dict['propietario']['iniciar sesión']}")
w print(f"Estrellas: {repo_dict['stargazers_count']}")
    imprimir(f"Repositorio: {repo_dict['html_url']}")
```

```
x print(f"Creado: {repo_dict['created_at']}")
y print(f"Actualizado: {repo_dict['updated_at']}")
  print(f"Descripción: {repo_dict['descripción']}")
```

Aquí imprimimos los valores para un número de claves del diccionario del primer repositorio. En u imprimimos el nombre del proyecto. Un diccionario completo representa al propietario del proyecto, por lo que en v usamos la clave de propietario para acceder al diccionario que representa al propietario y luego usamos la clave de inicio de sesión para obtener el nombre de inicio de sesión del propietario. En w imprimimos cuántas estrellas ha obtenido el proyecto y la URL del repositorio de GitHub del proyecto. Luego mostramos cuándo se creó x y cuándo se actualizó por última vez y. Finalmente, imprimimos la descripción del repositorio; la salida debería ser algo como esto:

```
Código de estado: 200
Repositorios totales: 3494032
Repositorios devueltos: 30

Información seleccionada sobre el primer repositorio:
Nombre: awesome-python
Dueño: vinta
Estrellas: 61549
Repositorio: https://github.com/vinta/awesome-python
Creado: 2014-06-27T21:00:06Z
Actualizado: 2019-02-17T04:30:00Z
Descripción: una lista seleccionada de increíbles marcos, bibliotecas y software de Python
    y recursos
```

Podemos ver que el proyecto de Python con más estrellas en GitHub a partir de este escrito es *awesome-python,* su propietario es el usuario *vinta,* y ha sido protagonizado por más de 60.000 usuarios de GitHub. Podemos ver la URL del repositorio del proyecto, su fecha de creación de junio de 2014 y que se actualizó recientemente. Además, la descripción nos dice que awesome-python contiene una lista de recursos populares de Python.

Resumen de los repositorios principales

Cuando hagamos una visualización de estos datos, querremos incluir más de un repositorio. Escribamos un bucle para imprimir la información seleccionada sobre cada repositorio que devuelve la llamada a la API para que podamos incluirlos todos en la visualización:

python_repos.py

```
--recorte--
# Explorar información sobre los repositorios.
repo_dicts = respuesta_dict['elementos']
print(f"Repositorios devueltos: {len(repo_dicts)}")

u print("\nInformación seleccionada sobre cada repositorio:")
v para repo_dict en repo_dicts:
      imprimir(f"\nNombre: {repo_dict['nombre']}")
      print(f"Propietario: {repo_dict['propietario']['iniciar sesión']}")
      print(f"Estrellas: {repo_dict['stargazers_count']}")
      imprimir(f"Repositorio: {repo_dict['html_url']}")
      print(f"Descripción: {repo_dict['descripción']}")
```

Imprimimos un mensaje de presentación en u. En v recorremos todos los diccionarios en repo_dicts. Dentro del ciclo, imprimimos el nombre de cada proyecto, su propietario, cuántas estrellas tiene, su URL en GitHub y la descripción del proyecto, como se muestra aquí:

```
Código de estado: 200
Repositorios totales: 3494040
Repositorios devueltos: 30

Información seleccionada sobre cada repositorio:

Nombre: impresionante-python
Dueño: vinta
Estrellas: 61549
Repositorio: https://github.com/vinta/awesome-python
Descripción: una lista seleccionada de increíbles marcos, bibliotecas y software de Python
    y recursos

Nombre: system-design-primer
Propietario: donnemartin
Estrellas: 57256
Repositorio: https://github.com/donnemartin/system-design-primer
Descripción: Aprende a diseñar sistemas a gran escala. preparación para el sistema
    entrevista de diseño. Incluye tarjetas flash de Anki.
--recorte--

Nombre: patrones-python
Dueño: faif
Estrellas: 19058
Repositorio: https://github.com/faif/python-patterns
Descripción: una colección de patrones de diseño/modismos en Python
```

Algunos proyectos interesantes aparecen en estos resultados, y podría valer la pena mirar algunos. Pero no pierda demasiado tiempo, porque en breve crearemos una visualización que hará que los resultados sean mucho más fáciles de leer.

Supervisión de los límites de velocidad de la API

La mayoría de las API tienen una tasa limitada, lo que significa que hay un límite en la cantidad de solicitudes que puede realizar en un período de tiempo determinado. Para ver si se está acercando a los límites de GitHub, ingrese *https:// api.github.com/ rate_limit* en un navegador web. Deberías ver una respuesta que comienza así:

```
{
    "recursos": {
        "centro": {
            "límite": 60,
            "restante": 58,
            "restablecer": 1550385312
        },
u "buscar": {
v "límite": 10,
w "restante": 8,
```

```
x           "restablecer": 1550381772
        },
        --recorte--
```

La información que nos interesa es el límite de velocidad para la búsqueda.

Ud. API Vemos en v que el límite es de 10 solicitudes por minuto y que nos quedan 8 solicitudes para el minuto actual w. El valor de reinicio representa el tiempo en *Unix* o tiempo de *época* (la cantidad de segundos desde la medianoche del 1 de enero de 1970) cuando nuestra cuota se reiniciará x. Si alcanza su cuota, recibirá una breve respuesta que le informará que ha alcanzado el límite de API.

Si alcanza el límite, simplemente espere hasta que se restablezca su cuota.

Muchas API requieren que se registre y obtenga una clave de API para realizar llamadas API. Al momento de escribir este artículo, GitHub no tiene tal requisito, pero si obtiene una clave API, sus límites serán mucho más altos.

Visualización de repositorios usando Plotly

Hagamos una visualización usando los datos que tenemos ahora para mostrar la popularidad relativa de los proyectos de Python en GitHub. Haremos un gráfico de barras interactivo: la altura de cada barra representará la cantidad de estrellas que el proyecto ha adquirido, y puede hacer clic en la etiqueta de la barra para ir a la página de inicio de ese proyecto en GitHub. Guarde una copia del programa en el que hemos estado trabajando como *python _repos_visual.py,* y luego modifíquelo para que quede como sigue:

python_repos _visual.py

```
solicitudes de importación

u de la barra de importación plotly.graph_objs
desde plotly importar sin conexión

v # Realice una llamada a la API y almacene la respuesta.
url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'
encabezados = {'Aceptar': 'aplicación/vnd.github.v3+json'}
r = solicitudes.get(url, encabezados=encabezados)
print(f"Código de estado: {r.status_code}")

# Resultados del proceso.
respuesta_dict = r.json()
repo_dicts = respuesta_dict['elementos']
w nombres_repo, estrellas = [], []
para repo_dict en repo_dicts:
    repo_names.append(repo_dict['nombre'])
    estrellas.append(repo_dict['stargazers_count'])

# Hacer visualización.
x datos = [{
    'tipo': 'barra',
    'x': nombres_repositorios,
    'y': estrellas,
}]
```

```
y mi_diseño = {
        'title': 'Proyectos de Python con más estrellas en GitHub',
        'xaxis': {'título': 'Repositorio'},
        'yaxis': {'title': 'Estrellas'},
    }

    fig = {'datos': datos, 'diseño': mi_diseño}
    offline.plot(fig, nombre de archivo='python_repos.html')
```

Importamos la clase Bar y el módulo fuera de línea de plotly u. nosotros no necesitamos importar la clase Diseño porque usaremos el enfoque de diccionario para definir el diseño, tal como lo hicimos para la lista de datos en el proyecto de ping del mapa de terremotos en el Capítulo 16. Continuamos imprimiendo el estado de la respuesta de llamada API para que sabremos si hay un problema v. También eliminamos parte del código que procesa la respuesta de la API, porque ya no estamos en la fase exploratoria; sabemos que tenemos los datos que queremos.

Luego creamos dos listas vacías w para almacenar los datos que incluiremos en el gráfico inicial. Necesitaremos el nombre de cada proyecto para etiquetar las barras y el número de estrellas para determinar la altura de las barras. En el bucle, agregamos el nombre de cada proyecto y la cantidad de estrellas que tiene a estas listas.

A continuación, definimos la lista de datos x. Este contiene un diccionario, como el que usamos en el Capítulo 16, que define el tipo de gráfico y proporciona los datos para los valores de x e y. Los valores de x son los nombres de los proyectos, y los valores de y son el número de estrellas que se le ha dado a cada proyecto.

En y definimos el diseño de este gráfico utilizando el enfoque de diccionario. En lugar de crear una instancia de la clase Layout , construimos un diccionario con las especificaciones de diseño que queremos usar. Establecemos un título para el gráfico general y definimos una etiqueta para cada eje.

La figura 17-1 muestra el gráfico resultante. Podemos ver que los primeros proyectos son significativamente más populares que el resto, pero todos ellos son proyectos importantes en el ecosistema de Python.

Figura 17-1: Los proyectos de Python con más estrellas en GitHub

Refinación de gráficos plotly

Vamos a refinar el estilo del gráfico. Como vio en el Capítulo 16, puede incluir todas las directivas de estilo como pares clave-valor en los diccionarios data y my_layout .

Los cambios en el objeto de datos afectan a las barras. Aquí hay una versión modificada del objeto de datos para nuestro gráfico que nos da un color específico y un borde claro para cada barra:

python_repos_visual.py

```
--recorte--
datos = [{
    'tipo': 'barra',
    'x': nombres_repositorios,
    'y': estrellas,
    'marcador': {
        'color': 'rgb(60, 100, 150)',
        'línea': {'ancho': 1.5, 'color': 'rgb(25, 25, 25)'}
    },
    'opacidad': 0.6,
}]
--recorte--
```

Los ajustes de marcador que se muestran aquí afectan el diseño de las barras. Establecemos un color azul personalizado para las barras y especificamos que se perfilarán con una línea gris oscuro de 1,5 píxeles de ancho. También establecemos la opacidad de las barras en 0,6 para suavizar un poco la apariencia del gráfico.

A continuación, modificaremos my_layout:

python_repos_visual.py

```
--recorte--
mi_diseño = {
    'title': 'Proyectos de Python con más estrellas en GitHub',
u 'fuente del título': {'tamaño': 28},
v 'ejex':'
        'título': 'Repositorio',
        'fuente del título': {'tamaño': 24},
        'tipo de letra': {'tamaño': 14},
    },
en 'yaxis': {
        'título': 'Estrellas',
        'fuente del título': {'tamaño': 24},
        'tipo de letra': {'tamaño': 14},
    },
}
--recorte--
```

Usamos la tecla 'titlefont' para definir el tamaño de fuente del título general del gráfico u. Dentro del diccionario 'xaxis' , agregamos configuraciones para controlar el tamaño de fuente del título del eje x ('titlefont') y también de las etiquetas de marca ('tickfont') v. Debido a que se trata de diccionarios anidados individuales, puede incluir claves para el color y la familia de fuentes de los títulos de los ejes y las etiquetas de marcas. En w definimos configuraciones similares para el eje y.

La figura 17-2 muestra el gráfico rediseñado.

Figura 17-2: Se ha refinado el estilo del gráfico.

Adición de información sobre

herramientas personalizada En Plotly, puede pasar el cursor sobre una barra individual para mostrar la información que representa la barra. Esto se conoce comúnmente como *información sobre herramientas* y, en este caso, actualmente muestra la cantidad de estrellas que tiene un proyecto. Vamos a crear una información sobre herramientas personalizada para mostrar la descripción de cada proyecto, así como el propietario del proyecto.

Necesitamos extraer algunos datos adicionales para generar la información sobre herramientas y modificar el objeto de datos :

python_repos_visual.py

```
--snip-- #
Resultados del
proceso. response_dict =
r.json() repo_dicts = response_dict['items']
u repo_names, stars, labels = [], [], [] for repo_dict
en repo_dicts: repo_names.append(repo_dict['name'])
        stars.append( repo_dict['recuento de
        observadores de estrellas'])

v propietario = repo_dict['propietario']['iniciar sesión']
        descripción = repo_dict['descripción'] w etiqueta
= f"{propietario}<br />{descripción}"
        etiquetas.append(etiqueta)


# Hacer visualización.
datos = [{ 'tipo': 'barra', 'x':
        nombres_repo, 'y':
        estrellas, x 'texto
        flotante': etiquetas,
'marcador': {
```

```
            'color': 'rgb(60, 100, 150)',
            'línea': {'ancho': 1.5, 'color': 'rgb(25, 25, 25)'}
        },
        'opacidad': 0.6,
}]
--recorte--
```

Primero definimos una nueva lista vacía, etiquetas, para contener el texto que queremos mostrar para cada proyecto u. En el bucle en el que procesamos los datos, extraemos el propietario y la descripción de cada proyecto v. Plotly le permite usar código HTML dentro de elementos de texto, por lo que generamos una cadena para la etiqueta con un salto de línea (
) entre el nombre de usuario del propietario del proyecto y la descripción w. Luego almacenamos esta etiqueta en las etiquetas de la lista .

En el diccionario de datos , agregamos una entrada con la clave 'hovertext' y asignamos es la lista que acabamos de crear x. A medida que Plotly crea cada barra, extraerá etiquetas de esta lista y solo las mostrará cuando el espectador se desplace sobre una barra.

La figura 17-3 muestra el gráfico resultante.

Figura 17-3: Al pasar el cursor sobre una barra, se muestra el propietario y la descripción del proyecto.

Agregar enlaces en los que se puede hacer clic a nuestro gráfico

Debido a que Plotly le permite usar HTML en elementos de texto, podemos agregar fácilmente enlaces a un gráfico. Usemos las etiquetas del eje x como una forma de permitir que el espectador visite la página de inicio de cualquier proyecto en GitHub. Necesitamos extraer las URL de los datos y usarlas al generar las etiquetas del eje x:

python_repos
_visual.py

```
--recorte--
# Resultados del proceso.
respuesta_dict = r.json()
repo_dicts = respuesta_dict['elementos']
```

```
u repo_links, estrellas, etiquetas = [], [], []
    para repo_dict en repo_dicts:
        repo_name = repo_dict['nombre']
v repo_url = repo_dict['html_url']
w repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
        repo_links.append(repo_link)

        estrellas.append(repo_dict['stargazers_count'])
        --recorte--

    # Hacer visualización.
    datos = [{
        'tipo': 'barra',
x 'x': repo_enlaces,
        'y': estrellas,
        --recorte--
    }]
    --recorte--
```

Actualizamos el nombre de la lista que estamos creando de repo_names a repo _enlaces para comunicar con mayor precisión el tipo de información que estamos recopilando para el gráfico u. Luego extraemos la URL del proyecto de repo_dict y la asignamos a la variable temporal repo_url v. En w generamos un enlace al proyecto. Usamos la etiqueta de anclaje HTML, que tiene la forma <a href='URL'>texto del enlace</ a>, para generar el enlace. Luego agregamos este enlace a la lista repo_links.

En x usamos esta lista para los valores de x en el gráfico. El resultado parece el igual que antes, pero ahora el espectador puede hacer clic en cualquiera de los nombres de proyectos en la parte inferior del gráfico para visitar la página de inicio de ese proyecto en GitHub. ¡Ahora tenemos una visualización interactiva e informativa de datos recuperados a través de una API!

Más sobre Plotly y la API de GitHub

Para leer más sobre cómo trabajar con gráficos de Plotly, hay dos buenos lugares para comenzar. Puede encontrar la *Guía del usuario de Plotly en Python* en *https:// plot.ly/ python / guía del usuario/.* Este recurso le brinda una mejor comprensión de cómo Plotly usa sus datos para construir una visualización y por qué aborda la definición de visualizaciones de datos de esta manera.

La *referencia de la figura de Python* en *https:// plot.ly/ python/ reference/* enumera todo el conjunto que puede usar para configurar visualizaciones de Plotly. Se enumeran todos los tipos de gráficos posibles, así como todos los atributos que puede establecer para cada opción de configuración.

Para obtener más información sobre la API de GitHub, consulte su documentación en *https:// desarrollador.github.com/ v3/.* Aquí aprenderá cómo obtener una amplia variedad de información específica de GitHub. Si tiene una cuenta de GitHub, puede trabajar con sus propios datos, así como con los datos disponibles públicamente para los repositorios de otros usuarios.

La API de noticias de hackers

Para explorar cómo usar las llamadas API en otros sitios, echemos un vistazo rápido a Hacker News *(http:// news.ycombinator.com/).* En Hacker News, las personas comparten artículos sobre programación y tecnología, y participan en animados debates sobre esos artículos. La API de Hacker News brinda acceso a datos sobre todos los envíos y comentarios en el sitio, y puede usar la API sin tener que registrarse para obtener una clave.

La siguiente llamada devuelve información sobre el artículo principal actual a partir de este escrito:

```
https://hacker-noticias.firebaseio.com/v0/item/19155826.json
```

Cuando ingrese esta URL en un navegador, verá que el texto de la página está encerrado entre llaves, lo que significa que es un diccionario. Pero la respuesta es difícil de examinar sin un formato mejor. Ejecutemos esta URL a través del método json.dump() , como hicimos en el proyecto del terremoto en el Capítulo 16, para que podamos explorar el tipo de información que se devuelve sobre un artículo:

hn_articulo.py

```
solicitudes de importación
importar json

# Realice una llamada a la API y almacene la respuesta.
url = 'https://hacker-news.firebaseio.com/v0/item/19155826.json'
r = solicitudes.obtener(url)
print(f"Código de estado: {r.status_code}")

# Explore la estructura de los datos.
respuesta_dict = r.json()
readable_file = 'datos/readable_hn_data.json'
con abierto (archivo_legible, 'w') como f:
    json.dump(response_dict, f, sangría=4)
```

Todo en este programa debería parecerle familiar, porque lo hemos usado todo en los dos capítulos anteriores. El resultado es un diccionario de información sobre el artículo con el ID 19155826:

legible_hn_datos.json

```
{
    "por": "jimktrains2",
u "descendientes": 220,
    "identificación": 19155826,
v "niños": [
        19156572,
        19158857,
        --recorte--
    ],
    "puntuación": 722,
    "tiempo": 1550085414,
w "title": "Oportunidad Mars Rover de la NASA concluye una misión de 15 años",
    "tipo": "historia",
x "url": "https://www.nytimes.com/.../mars-opportunity-rover-dead.html"
}
```

El diccionario contiene una serie de claves con las que podemos trabajar. La clave 'descendientes' nos dice el número de comentarios que ha recibido el artículo u. La clave 'niños' proporciona los ID de todos los comentarios realizados directamente en respuesta a este envío v. Cada uno de estos comentarios también puede tener sus propios comentarios, por lo que la cantidad de descendientes que tiene un envío suele ser mayor que la cantidad de niños. Podemos ver el título del artículo que se está discutiendo w, y una URL para el artículo que se está discutiendo también x.

La siguiente URL devuelve una lista simple de todos los ID de la parte superior actual artículos en Hacker News:

```
https://hacker-news.firebaseio.com/v0/topstories.json
```

Podemos usar esta llamada para averiguar qué artículos están en la página de inicio en este momento y luego generar una serie de llamadas API similares a la que acabamos de examinar. Con este enfoque, podemos imprimir un resumen de todos los artículos en la portada de Hacker News en este momento:

hn_submissions.py

```
desde el operador import itemgetter

solicitudes de importación

# Realice una llamada a la API y almacene la respuesta.
u url = 'https://hacker-news.firebaseio.com/v0/topstories.json'
r = solicitudes.obtener(url)
print(f"Código de estado: {r.status_code}")

# Procesar información sobre cada envío.
v id_sumisión = r.json()
w dictados_sumisión = []
para id_presentación en id_presentación[:30]:
    # Realice una llamada a la API por separado para cada envío.
x url = f"https://hacker-news.firebaseio.com/v0/item/{submission_id}.json"
    r = solicitudes.obtener(url)
    print(f"id: {submission_id}\tstatus: {r.status_code}")
    respuesta_dict = r.json()

    # Cree un diccionario para cada artículo.
y sumisión_dict = {
        'título': response_dict['título'],
        'hn_link': f"http://news.ycombinator.com/item?id={submission_id}",
        'comentarios': response_dict['descendientes'],
    }
z submission_dicts.append(submission_dict)

{ submission_dicts = sorted(submission_dicts, key=itemgetter('comentarios'),
                            inverso = Verdadero)

| para dict_presentación en dict_presentación:
    imprimir(f"\nTítulo: {submission_dict['título']}")
    print(f"Enlace de discusión: {submission_dict['hn_link']}")
    print(f"Comentarios: {submission_dict['comentarios']}")
```

Primero, hacemos una llamada a la API y luego imprimimos el estado de la respuesta u. Esta llamada API devuelve una lista que contiene los ID de hasta los 500 artículos más populares en Hacker News en el momento en que se emite la llamada. Luego, convertimos el objeto de respuesta en una lista de Python en v, que almacenamos en submit_ids. Usaremos estos ID para crear un conjunto de diccionarios que almacenen información sobre uno de los envíos actuales.

Configuramos una lista vacía llamada submit_dicts en w para almacenar esta dicción Aries. Luego repasamos los ID de las 30 presentaciones principales. Realizamos una nueva llamada a la API para cada envío generando una URL que incluye el valor actual de submit_id x. Imprimimos el estado de cada solicitud junto con su ID, para que podamos ver si tiene éxito.

En y creamos un diccionario para el envío que se está procesando actualmente, donde almacenamos el título del envío, un enlace a la página de discusión de ese elemento y la cantidad de comentarios que el artículo ha recibido hasta el momento. Luego añadimos cada dict_submisión a la lista dict_submisión z.

Cada presentación en Hacker News se clasifica de acuerdo con una puntuación general basada en una serie de factores que incluyen cuántas veces se votó, cuántos comentarios se recibieron y qué tan reciente es la presentación. Queremos ordenar la lista de diccionarios por el número de comentarios. Para hacer esto, usamos una función llamada itemgetter() {, que proviene del módulo operator . Pasamos a esta función la clave 'comentarios', y extrae el valor asociado con esa clave de cada diccionario en la lista. El ordenado () Luego, la función usa este valor como base para ordenar la lista. Ordenamos la lista en orden inverso para colocar primero las historias más comentadas.

Una vez ordenada la lista, recorremos la lista en | e imprima tres piezas de información sobre cada uno de los principales envíos: el título, un enlace a la página de discusión y la cantidad de comentarios que tiene actualmente el envío:

```
Código de estado: 200
identificación: 19155826 estado: 200
id: 19180181 id:        estado: 200
19181473 --             estado: 200
recorte--

Título: La oportunidad Mars Rover de la NASA concluye una misión de 15 años
Enlace de discusión: http://news.ycombinator.com/item?id=19155826
Comentarios: 220

Título: Pregúntele a HN: ¿Es práctico crear un modelo de cohete controlado por software?
Enlace de discusión: http://news.ycombinator.com/item?id=19180181
Comentarios: 72

Título: Hacer mi propio teclado USB desde cero
Enlace de discusión: http://news.ycombinator.com/item?id=19181473
Comentarios: 62
--recorte--
```

Usaría un proceso similar para acceder y analizar información con cualquier API. Con estos datos, puede hacer una visualización que muestre qué envíos han inspirado las discusiones recientes más activas. Esta es también la base de las aplicaciones que brindan una experiencia de lectura personalizada para sitios como Hacker News. Para obtener más información sobre el tipo de información a la que puede acceder a través de la API de Hacker News, visite la página de documentación en *https:// github .com/ HackerNews/ API/.*

Inténtalo tú mismo

17-1. Otros idiomas: modifique la llamada a la API en python_repos.py para que genere un gráfico que muestre los proyectos más populares en otros idiomas. Pruebe lenguajes como JavaScript, Ruby, C, Java, Perl, Haskell y Go.

17-2. Discusiones activas: utilizando los datos de hn_submissions.py, cree un gráfico de barras que muestre las discusiones más activas que se están produciendo actualmente en Hacker News. La altura de cada barra debe corresponder a la cantidad de comentarios que tiene cada envío. La etiqueta de cada barra debe incluir el título del envío y debe actuar como un enlace a la página de discusión de ese envío.

17-3. Probando python_repos.py: en python_repos.py, imprimimos el valor de status_code para asegurarnos de que la llamada API fue exitosa. Escriba un programa llamado test_python_repos.py que use unittest para afirmar que el valor de status_code es 200. Calcule otras afirmaciones que pueda hacer, por ejemplo, que se espera la cantidad de elementos devueltos y que la cantidad total de depósitos es mayor que cierta cantidad.

17-4. Exploración adicional: visite la documentación de Plotly y la API de GitHub o la API de Hacker News. Utilice parte de la información que encuentre allí para personalizar el estilo de los gráficos que ya hemos creado o extraiga información diferente y cree sus propias visualizaciones.

Resumen

En este capítulo, aprendió a usar las API para escribir programas independientes que recopilan automáticamente los datos que necesitan y usan esos datos para crear una visualización. Usó la API de GitHub para explorar los proyectos de Python con más estrellas en GitHub y también examinó brevemente la API de Hacker News. Aprendió a usar el paquete Requests para emitir automáticamente una llamada API a GitHub y cómo procesar los resultados de esa llamada. También se introdujeron algunas configuraciones de Plotly que personalizan aún más la apariencia de los gráficos que genera.

En el próximo capítulo, utilizará Django para crear una aplicación web como proyecto final.



Proyecto 3

Aplicaciones web



18

Primeros pasos con Django

Detrás de escena, los sitios web de hoy son aplicaciones ricas que actúan como aplicaciones de escritorio completamente desarrolladas. Python tiene un gran conjunto de herramientas llamado Django para crear aplicaciones web. Django es un *marco web: un* conjunto de herramientas diseñadas para ayudarlo a crear sitios web interactivos. En este capítulo, aprenderá a usar Django *(https:// djangoproject.com/)* para crear un proyecto llamado Registro de aprendizaje, un sistema de diario en línea que le permite realizar un seguimiento de la información que ha aprendido sobre temas particulares.

Escribiremos una especificación para este proyecto y luego definiremos modelos para los datos con los que trabajará la aplicación. Usaremos el sistema de administración de Django para ingresar algunos datos iniciales y luego aprenderá a escribir vistas y plantillas para que Django pueda construir las páginas del sitio.

Django puede responder a solicitudes de página y facilitar la lectura y escritura en una base de datos, administrar usuarios y mucho más. En los Capítulos 19 y 20, refinará el proyecto Registro de aprendizaje y luego lo implementará en un servidor en vivo para que usted (y sus amigos) puedan usarlo.

Configuración de un proyecto

Al comenzar un proyecto, primero debe describir el proyecto en una especificación o *especificación.* A continuación, configurará un entorno virtual en el que compilar el proyecto.

Escribir una especificación

Una especificación completa detalla los objetivos del proyecto, describe la funcionalidad del proyecto y analiza su apariencia e interfaz de usuario. Como cualquier buen proyecto o plan de negocios, una especificación debe mantenerlo enfocado y ayudarlo a mantener su proyecto encaminado. No escribiremos una especificación completa del proyecto aquí, pero presentaremos algunos objetivos claros para mantener el proceso de desarrollo enfocado. Aquí está la especificación que usaremos:

> Escribiremos una aplicación web llamada Registro de aprendizaje que permita a los usuarios registrar los temas que les interesan y hacer entradas en el diario a medida que aprenden sobre cada tema. La página de inicio del Registro de aprendizaje describirá el sitio e invitará a los usuarios a registrarse o iniciar sesión. Una vez que haya iniciado sesión, un usuario puede crear nuevos temas, agregar nuevas entradas y leer y editar entradas existentes.

Cuando aprende sobre un tema nuevo, llevar un diario de lo que ha aprendido puede ser útil para rastrear y revisar la información. Una buena aplicación hace que este proceso sea eficiente.

Creación de un entorno virtual

Para trabajar con Django, primero configuraremos un entorno virtual. Un *entorno virtual* es un lugar en su sistema donde puede instalar paquetes y aislarlos de todos los demás paquetes de Python. Separar las bibliotecas de un proyecto de otros proyectos es beneficioso y será necesario cuando implementemos Learning Log en un servidor en el Capítulo 20.

Cree un nuevo directorio para su proyecto llamado *learning_log,* cambie a ese directorio en una terminal e ingrese el siguiente código para crear un entorno virtual:

```
aprendizaje_log$ python -m venv ll_env
aprendizaje_log$
```

Aquí estamos ejecutando el módulo de entorno virtual venv y usándolo para crear un entorno virtual llamado *ll_env* (tenga en cuenta que esto es *ll_env* con dos L minúsculas , no dos). Si usa un comando como **python3**
cuando ejecute programas o instale paquetes, asegúrese de usar ese comando aquí.

Activación del entorno virtual

Ahora necesitamos activar el entorno virtual usando el siguiente comando:

```
learning_log$ fuente ll_env/bin/activar
u (ll_env)learning_log$
```

Este comando ejecuta el script *active* en *ll_env/ bin.* Cuando el entorno está activo, verá el nombre del entorno entre paréntesis, como se muestra en u; luego puede instalar paquetes en el entorno y usar paquetes que ya se han instalado. Paquetes que instalas en *ll_env* estará disponible solo mientras el entorno esté activo.

Si utiliza Windows, utilice el comando ll_env\Scripts\activate (sin la palabra fuente) para activar el entorno virtual. Si usa PowerShell, es posible que deba poner en mayúscula Activar.

Para dejar de usar un entorno virtual, ingrese **desactivar:**

```
(ll_env)learning_log$ desactivar
aprendizaje_log$
```

El entorno también se volverá inactivo cuando cierre la terminal en la que se está ejecutando.

Instalando Django

Una vez activado el entorno virtual, ingrese lo siguiente para instalar Django:

```
(ll_env)learning_log$ pip instalar django
coleccionando django
--recorte--
Instalación de paquetes recopilados: pytz, django
Django-2.2.0 instalado con éxito pytz-2018.9 sqlparse-0.2.4
(ll_env)registro_de_aprendizaje$
```

Debido a que estamos trabajando en un entorno virtual, que es su propio entorno autónomo, este comando es el mismo en todos los sistemas. No hay necesidad de usar el indicador --user , y no hay necesidad de usar comandos más largos, como python -m pip install *nombre_paquete.*

Tenga en cuenta que Django estará disponible solo cuando el entorno *ll_env* esté activo.

Django lanza una nueva versión aproximadamente cada ocho meses, por lo que es posible que vea una versión más nueva cuando instale Django. Lo más probable es que este proyecto funcione como está escrito aquí, incluso en las versiones más nuevas de Django. Si quiere asegurarse de usar la misma versión de Django que ve aquí, use el comando pip install django==2.2.. Esto instalará la última versión de Django 2.2. Si tiene algún problema relacionado con la versión que está utilizando, consulte los recursos en línea para el libro en* https://nostarch.com/pythoncrashcourse2e/.

Crear un proyecto en Django

Sin salir del entorno virtual activo (recuerda buscar *ll_env* entre paréntesis en el indicador del terminal), ingrese los siguientes comandos para crear un nuevo proyecto:

```
u (ll_env)learning_log$ django-admin startproject learning_log .
v (ll_env)learning_log$ ls
  registro_de_aprendizaje ll_env administrar.py
w (ll_env)registro_de_aprendizaje$ ls registro_de_aprendizaje
  __init__.py configuración.py urls.py wsgi.py
```

El comando en u le dice a Django que configure un nuevo proyecto llamado *aprendizaje _Iniciar sesión.* El punto al final del comando crea el nuevo proyecto con una estructura de directorios que facilitará la implementación de la aplicación en un servidor cuando terminemos de desarrollarla.

No olvide este punto, o puede encontrarse con algunos problemas de configuración cuando implemente la aplicación. Si olvida el punto, elimine los archivos y carpetas que se crearon (excepto ll_env) *y ejecute el comando nuevamente.*

Ejecutar el comando ls (dir en Windows) v muestra que Django ha creado un nuevo directorio llamado *learning_log.* También creó un archivo *manage.py* , que es un programa corto que recibe comandos y los alimenta a la parte relevante de Django para ejecutarlos. Usaremos estos comandos para administrar tareas, como trabajar con bases de datos y ejecutar servidores.

El directorio *learning_log* contiene cuatro archivos w; los más importantes son *settings.py, urls.py* y *wsgi.py.* El archivo *settings.py* controla cómo Django interactúa con su sistema y administra su proyecto. Modificaremos algunas de estas configuraciones y agregaremos algunas configuraciones propias a medida que evolucione el proyecto. Las *urls.py* El archivo le dice a Django qué páginas construir en respuesta a las solicitudes del navegador. El archivo *wsgi.py* ayuda a Django a servir los archivos que crea. El nombre del archivo es un acrónimo de *interfaz de puerta de enlace del servidor web.*

Crear la base de datos

Django almacena la mayor parte de la información de un proyecto en una base de datos, por lo que a continuación debemos crear una base de datos con la que Django pueda trabajar. Ingrese el siguiente comando (todavía en un entorno activo):

```
  (ll_env)learning_log$ python manage.py migrar
u Operaciones a realizar: aplicar
    todas las migraciones: administración, autenticación, tipos de contenido, sesiones
  Ejecutando migraciones:
    Aplicando contenttypes.0001_initial... Aceptar
    Aplicando auth.0001_initial... OK
    --recorte--
    Aplicando sesiones.0001_inicial... OK v
(ll_env)learning_log$ ls
  db.sqlite3 learning_log ll_env manage.py
```

Cada vez que modificamos una base de datos, decimos que estamos *migrando* la base de datos. Emitir el comando de migración por primera vez le dice a Django que se asegure de que la base de datos coincida con el estado actual del proyecto. La primera vez que ejecutamos este comando en un nuevo proyecto usando SQLite (más sobre SQLite en un momento), Django creará una nueva base de datos para nosotros. En u, Django informa que preparará la base de datos para almacenar la información que necesita para manejar las tareas administrativas y de autenticación.

Ejecutar el comando ls muestra que Django creó otro archivo llamado *db.sqlite3* v. SQLite es una base de datos que se ejecuta en un solo archivo; es ideal para escribir aplicaciones sencillas porque no tendrá que prestar mucha atención a la gestión de la base de datos.

En un entorno virtual activo, use el comando python para ejecutar manage.py com mands, incluso si usa algo diferente, como python3, para ejecutar otros programas. En un entorno virtual, el comando python hace referencia a la versión de Python que creó el entorno virtual.

Ver el proyecto

Asegurémonos de que Django haya configurado el proyecto correctamente. Ingrese el comando runserver de la siguiente manera para ver el proyecto en su estado actual:

```
(ll_env)learning_log$ python manage.py runserver
Watchman no disponible: pywatchman no instalado.
Observando los cambios de archivos con StatReloader
Realizando comprobaciones del sistema...

u La verificación del sistema no identificó problemas (0 silenciados).
  18 febrero 2019 - 16:26:07
v Django versión 2.2.0, utilizando la configuración 'learning_log.settings'
w Iniciando el servidor de desarrollo en http://127.0.0.1:8000/
  Salga del servidor con CONTROL-C.
```

Django debería iniciar un servidor llamado servidor de *desarrollo,* para que pueda ver el proyecto en su sistema y ver qué tan bien funciona. Cuando solicita una página ingresando una URL en un navegador, el servidor Django responde a esa solicitud creando la página adecuada y enviándola al navegador.

En u, Django verifica para asegurarse de que el proyecto esté configurado correctamente; en v informa la versión de Django en uso y el nombre del archivo de configuración en uso; y en w informa la URL donde se está sirviendo el proyecto. La URL *http:// 127.0.0.1:8000/* indica que el proyecto está escuchando solicitudes en el puerto 8000 de su computadora, que se denomina host local. El término *host local* se refiere a un servidor que solo procesa solicitudes en su sistema; no permite que nadie más vea las páginas que está desarrollando.

Abra un navegador web e ingrese la URL *http:// localhost:8000/,* o *http:// 127.0.0.1:8000/* si el primero no funciona. Debería ver algo como la Figura 18-1, una página que crea Django para informarle que todo funciona correctamente hasta el momento. Mantenga el servidor funcionando por ahora, pero cuando desee detener el servidor, presione ctrl-C en la terminal donde se emitió el comando runserver .

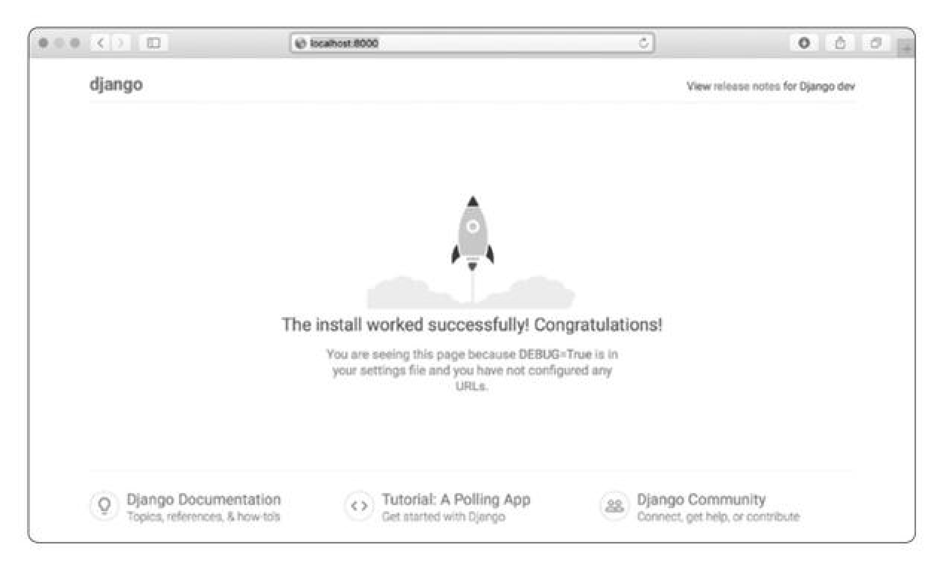


Figura 18-1: Todo funciona hasta ahora.

N ota *ecibe el mensaje de error* Ese puerto ya está en uso, *dígale a Django que use un puerto diferente ingresando python manage.py runserver 8001, y luego recorra los números más altos hasta que encuentre un puerto abierto.*

Inténtalo tú mismo

18-1. Nuevos proyectos: para tener una mejor idea de lo que hace Django, cree un par de proyectos vacíos y observe lo que crea Django. Cree una nueva carpeta con un nombre simple, como snap_gram o insta_chat (fuera de su directorio learning_log), navegue hasta esa carpeta en una terminal y cree un entorno virtual. Instale Django y ejecute el comando django-admin.py startproject snap_gram. (asegúrese de incluir el punto al final del comando).

Mire los archivos y carpetas que crea este comando y compárelos con el registro de aprendizaje. Haga esto varias veces hasta que esté familiarizado con lo que crea Django al iniciar un nuevo proyecto. Luego elimine los directorios del proyecto si lo desea.

Iniciar una aplicación

Un *proyecto* de Django se organiza como un grupo de *aplicaciones* individuales que funcionan juntas para que el proyecto funcione como un todo. Por ahora, crearemos una sola aplicación para hacer la mayor parte del trabajo de nuestro proyecto. Agregaremos otra aplicación en el Capítulo 19 para administrar cuentas de usuario.

Debe dejar el servidor de desarrollo ejecutándose en la ventana de terminal que abrió anteriormente. Abra una nueva ventana (o pestaña) de terminal y navegue hasta el directorio que contiene *manage.py.* Active el entorno virtual y luego ejecute el comando startapp :

```
learning_log$ fuente ll_env/bin/activar
(ll_env)learning_log$ python manage.py startapp learning_logs
u (ll_env)learning_log$ ls
db.sqlite3 learning_log learning_logs ll_env manage.py
v (ll_env)registro_de_aprendizaje$ ls registros_de_aprendizaje/
__init__.py admin.py apps.py migraciones models.py tests.py views.py
```

El comando startapp *appname* le dice a Django que cree la infraestructura necesaria para construir una aplicación. Cuando busque en el directorio del proyecto ahora, verá una nueva carpeta llamada *learning_logs* u. Abra esa carpeta para ver qué ha creado Django v. Los archivos más importantes son *models.py, admin.py* y *views.py.* Usaremos *models.py* para definir los datos que queremos administrar en nuestra aplicación. Veremos *admin.py* y *views.py* un poco más tarde.

Definición de modelos

Pensemos en nuestros datos por un momento. Cada usuario deberá crear una serie de temas en su registro de aprendizaje. Cada entrada que hagan estará vinculada a un tema, y estas entradas se mostrarán como texto. También necesitaremos almacenar la marca de tiempo de cada entrada, para que podamos mostrar a los usuarios cuándo hicieron cada entrada.

Abra el archivo *models.py* y observe su contenido existente:

modelos.py

```
desde modelos de importación django.db

# Crea tus modelos aquí.
```

Se está importando un módulo llamado modelos y se nos invita a crear nuestros propios modelos. Un *modelo* le dice a Django cómo trabajar con los datos que se almacenarán en la aplicación. En cuanto al código, un modelo es solo una clase; tiene atributos y métodos, como todas las clases que hemos discutido. Este es el modelo de los temas que almacenarán los usuarios:

```
desde modelos de importación django.db

Tema de clase (modelos.Modelo):
    """Un tema sobre el que el usuario está aprendiendo."""
u texto = modelos.CharField(max_length=200)
v date_added = modelos.DateTimeField(auto_now_add=True)

w def __str__(uno mismo):
        """Retorna una representación de cadena del modelo."""
        devolver auto.texto
```

Hemos creado una clase llamada Tema, que hereda de Modelo, una clase principal incluida en Django que define la funcionalidad básica de un modelo. Agregamos dos atributos a la clase Topic : text y date_added.

El atributo de texto es un CharField, un dato compuesto por caracteres o texto u. Utiliza CharField cuando desea almacenar una pequeña cantidad de texto, como un nombre, un título o una ciudad. Cuando definimos un atributo CharField , debemos decirle a Django cuánto espacio debe reservar en la base de datos. Aquí le damos una longitud máxima de 200 caracteres, que debería ser suficiente para contener la mayoría de los nombres de temas.

El atributo date_added es un DateTimeField, un dato que registrará una fecha y hora v. Pasamos el argumento auto_now_add=True, que le dice a Django que establezca automáticamente este atributo en la fecha y hora actual siempre que el usuario cree un nuevo tema.

Para ver los diferentes tipos de campos que puede usar en un modelo, consulte la Referencia de campo del modelo de Django *en* https://docs.djangoproject.com/en/2.2/ref/models /los campos/. *No necesitará toda la información en este momento, pero será extremadamente útil cuando esté desarrollando sus propias aplicaciones.*

Le decimos a Django qué atributo usar por defecto cuando muestra información sobre un tema. Django llama a un método __str__() para mostrar una representación simple de un modelo. Aquí hemos escrito un método __str__() que devuelve la cadena almacenada en el atributo de texto w.

Activación de modelos

Para usar nuestros modelos, debemos decirle a Django que incluya nuestra aplicación en el proyecto general. Abra *settings.py* (en el directorio *learning_log/ learning_log);* verás una sección que le dice a Django qué aplicaciones están instaladas y funcionan juntas en el proyecto:

configuración.py

```
--recorte--
APLICACIONES_INSTALADAS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.archivos estáticos',
]
--recorte--
```

Agregue nuestra aplicación a esta lista modificando INSTALLED_APPS para que se vea así:

```
--recorte--
APLICACIONES_INSTALADAS = [
    # Mis aplicaciones
    'registro_de_aprendizaje',

   # Aplicaciones de django predeterminadas.
    'django.contrib.admin',
```

```
    --recorte--
]
--recorte--
```

Agrupar aplicaciones en un proyecto ayuda a realizar un seguimiento de ellas a medida que el proyecto crece para incluir más aplicaciones. Aquí comenzamos una sección llamada *Mis aplicaciones,* que incluye solo learning_logs por ahora. Es importante colocar sus propias aplicaciones antes de las aplicaciones predeterminadas en caso de que necesite anular cualquier comportamiento de las aplicaciones predeterminadas con su propio comportamiento personalizado.

A continuación, debemos decirle a Django que modifique la base de datos para que pueda almacenar información relacionada con el tema del modelo. Desde la terminal, ejecuta el siguiente comando:

```
(ll_env)learning_log$ python manage.py makemigrations learning_logs
Migraciones para 'learning_logs':
  registros_de_aprendizaje/migraciones/0001_inicial.py
    - Crear tema modelo
(ll_env)registro_de_aprendizaje$
```

El comando makemigrations le dice a Django que descubra cómo modificar la base de datos para que pueda almacenar los datos asociados con cualquier modelo nuevo que hayamos definido. El resultado aquí muestra que Django ha creado un archivo de migración llamado *0001_initial.py.* Esta migración creará una tabla para el modelo Tema en la base de datos

Ahora aplicaremos esta migración y Django modificará la base de datos. para nosotros:

```
(ll_env)learning_log$ python manage.py migrar
Operaciones a realizar:
  Aplicar todas las migraciones: administración, autenticación, tipos de contenido, registros de aprendizaje, sesiones
Ejecutando migraciones:
u Aplicando learning_logs.0001_initial... OK
```

La mayor parte de la salida de este comando es idéntica a la primera vez que emitimos el comando de migración . La línea que debemos verificar aparece en u, donde Django confirma que la migración de learning_logs funcionó bien.

Siempre que queramos modificar los datos que maneja Learning Log, seguiremos estos tres pasos: modificar *models.py,* llamar a makemigrations en learning_logs y decirle a Django que migre el proyecto.

El sitio de administración de Django

Django facilita el trabajo con sus modelos a través del *sitio de administración.* Solo los administradores del sitio usan el sitio de administración, no los usuarios generales. En esta sección, configuraremos el sitio de administración y lo usaremos para agregar algunos temas a través del modelo de temas .

Configuración de un superusuario

Django le permite crear un *superusuario,* un usuario que tiene todos los privilegios disponibles en el sitio. Los *privilegios* de un usuario controlan las acciones que el usuario puede realizar.

La configuración de privilegios más restrictiva permite que un usuario solo lea información pública en el sitio. Los usuarios registrados suelen tener el privilegio de leer sus propios datos privados y cierta información seleccionada disponible solo para los miembros. Para administrar de manera efectiva una aplicación web, el propietario del sitio generalmente necesita acceso a toda la información almacenada en el sitio. Un buen administrador tiene cuidado con la información confidencial de sus usuarios, porque los usuarios confían mucho en las aplicaciones a las que acceden.

Para crear un superusuario en Django, ingrese el siguiente comando y responder a las indicaciones:

```
(ll_env)learning_log$ python manage.py createsuperuser
u Nombre de usuario (déjelo en blanco para usar 'eric'): ll_admin
v Dirección de correo electrónico:
w Contraseña:
Contraseña de nuevo):
Superusuario creado con éxito.
(ll_env)registro_de_aprendizaje$
```

Cuando ejecuta el comando createsuperuser, Django le pide que ingrese un nombre de usuario para el superusuario u. Aquí estoy usando *ll_admin,* pero puede ingresar cualquier nombre de usuario que desee. Puede ingresar una dirección de correo electrónico si lo desea o simplemente dejar este campo en blanco v. Deberá ingresar su contraseña dos veces w.

Cierta información confidencial se puede ocultar a los administradores de un sitio. Por ejemplo, Django no almacena la contraseña que ingresa; en su lugar, almacena una cadena derivada de la contraseña, llamada hash. *Cada vez que ingresa su contraseña, Django procesa su entrada y la compara con el hash almacenado. Si los dos hashes coinciden, estás autenticado. Al exigir que los hashes coincidan, si un atacante obtiene acceso a la base de datos de un sitio, podrá leer los hashes almacenados, pero no las contraseñas. Cuando un sitio está configurado correctamente, es casi imposible obtener las contraseñas originales de los hashes.*

Registro de un modelo con el sitio de administración

Django incluye algunos modelos en el sitio de administración automáticamente, como Usuario y Group, pero los modelos que creamos deben agregarse manualmente.

Cuando iniciamos la aplicación learning_logs , Django creó un archivo *admin.py* en el mismo directorio que *models.py.* Abra el archivo *admin.py :*

administrador.py

```
desde django.contrib administrador de importación

# Registre sus modelos aquí.
```

Para registrar Tema con el sitio de administración, ingrese lo siguiente:

```
desde django.contrib administrador de importación

u del tema de importación de .models

v admin.site.register(Tema)
```

Este código primero importa el modelo que queremos registrar, Tema u. El punto delante de models le dice a Django que busque *models.py* en el mismo directorio que *admin.py.* El código admin.site.register() le dice a Django que administre nuestro modelo a través del sitio de administración v.

Ahora use la cuenta de superusuario para acceder al sitio de administración. Ir a *http:// localhost:8000/ admin/,* e ingrese el nombre de usuario y la contraseña del superusuario que acaba de crear. Debería ver una pantalla como la de la Figura 18-2.

Esta página le permite agregar nuevos usuarios y grupos, y cambiar los existentes. También puede trabajar con datos relacionados con el modelo de tema que acabamos de definir.

Figura 18-2: El sitio de administración con *Tema* incluido

Si ve un mensaje en su navegador que indica que la página web no está disponible, asegúrese de que todavía tiene el servidor Django ejecutándose en una ventana de terminal. Si no lo hace, active un entorno virtual y vuelva a emitir el comando python manage.py runserver. Si tiene problemas para ver su proyecto en cualquier punto del proceso de desarrollo, cerrar cualquier terminal abierta y volver a ejecutar el comando *runserver es un buen primer paso para solucionar el problema.*

Agregar temas

Ahora que el tema se ha registrado en el sitio de administración, agreguemos nuestro primer tema. Haga clic en **Temas** para ir a la página Temas, que está casi vacía porque todavía no tenemos temas para administrar. Haga clic en **Agregar tema** y aparecerá un formulario para agregar un nuevo tema. Ingrese **Ajedrez** en el primer cuadro y haga clic en **Guardar.** Volverá a la página de administración de Temas y verá el tema que acaba de crear.

Vamos a crear un segundo tema para tener más datos con los que trabajar. Haga clic en **Agregar tema** nuevamente e ingrese **Escalada en roca.** Haga clic en **Guardar** y volverá a la página principal de Temas. Ahora verá Ajedrez y Escalada en roca en la lista.

Definición del modelo de entrada

Para que un usuario registre lo que ha estado aprendiendo sobre ajedrez y escalada en roca, necesitamos definir un modelo para los tipos de entradas que los usuarios pueden hacer en sus registros de aprendizaje. Cada entrada debe estar asociada con un tema en particular. Esta relación se denomina relación de *muchos a uno,* lo que significa que muchas entradas se pueden asociar con un tema.

Aquí está el código para el modelo Entry . Colóquelo en su archivo *models.py :*

modelos.py

```
desde modelos de importación django.db

Tema de clase (modelos.Modelo):
        --recorte--

Entrada de clase u (modelos.Modelo):
        """Algo específico aprendido sobre un tema"."""
v topic = models.ForeignKey(Tema, on_delete=models.CASCADE)
w texto = modelos.TextField()
        date_added = modelos.DateTimeField(auto_now_add=True)

Metaclase x:
                verbose_name_plural = 'entradas'

        def __str__(uno mismo):
                """Retorna una representación de cadena del modelo."""
y               devuelve f"{self.text[:50]}..."
```

La clase Entry hereda de la clase Model base de Django , al igual que Topic lo hiciste. El primer atributo, tema, es una instancia de ForeignKey v. Una *clave externa* es un término de base de datos; es una referencia a otro registro en la base de datos. Este es el código que conecta cada entrada a un tema específico. A cada tema se le asigna una clave, o ID, cuando se crea. Cuando Django necesita establecer una conexión entre dos datos, utiliza la clave asociada con cada dato. Usaremos estas conexiones en breve para recuperar todas las entradas asociadas con un tema determinado. El argumento on_delete=models.CASCADE le dice a Django que cuando se elimina un tema, todas las entradas asociadas con ese tema también deben eliminarse. Esto se conoce como *eliminación en cascada.*

El siguiente es un atributo llamado texto, que es una instancia de TextField w. Este tipo de campo no necesita un límite de tamaño, porque no queremos limitar el tamaño de las entradas individuales. El atributo date_added nos permite presentar las entradas en el orden en que fueron creadas y colocar una marca de tiempo junto a cada entrada.

En x anidamos la clase Meta dentro de nuestra clase Entry . La clase Meta contiene información adicional para administrar un modelo; aquí, nos permite establecer un atributo especial que le dice a Django que use *Entradas* cuando necesita referirse a más de una entrada. Sin esto, Django se referiría a múltiples entradas como *Entradas.*

El método __str__() le dice a Django qué información mostrar cuando se refiere a entradas individuales. Debido a que una entrada puede ser un cuerpo largo de texto, le decimos a Django que muestre solo los primeros 50 caracteres del texto y. También agregamos puntos suspensivos para aclarar que no siempre mostramos la entrada completa.

Migración del modelo de entrada

Debido a que agregamos un nuevo modelo, necesitamos migrar la base de datos nuevamente. Este proceso se volverá bastante familiar: modifica *models.py,* ejecuta el comando python manage.py makemigrations *app_name* y luego ejecuta el comando python manage.py migrate.

Migre la base de datos y verifique el resultado ingresando lo siguiente comandos:

```
(ll_env)learning_log$ python manage.py makemigrations learning_logs
Migraciones para 'learning_logs':
u learning_logs/migrations/0002_entry.py
    - Crear entrada modelo
(ll_env)learning_log$ python manage.py migrar
Operaciones a realizar: --
  snip--
v Aplicando learning_logs.0002_entry... Aceptar
```

Se genera una nueva migración llamada *0002_entry.py* , que le dice a Django cómo modificar la base de datos para almacenar información relacionada con el modelo Entrada u. Cuando emitimos el comando de migración , vemos que Django aplicó esta migración y todo estuvo bien v.

Registro de entrada con el sitio de administración

También necesitamos registrar el modelo Entry . Así es como debería verse *admin.py* ahora:

administrador.py

```
desde django.contrib administrador de importación

de .modelos importar tema, entrada

admin.site.register(Tema)
admin.sitio.registrar(Entrada)
```

Vuelva a *http:// localhost/ admin/* y debería ver las *entradas* enumeradas en *Learning_Logs.* Haga clic en el enlace **Agregar** para Entradas, o haga clic en **Entradas** y luego seleccione **Agregar entrada.** Debería ver una lista desplegable para seleccionar el tema para el que está creando una entrada y un cuadro de texto para agregar una entrada. Seleccione **Ajedrez** de la lista desplegable y agregue una entrada. Aquí está la primera entrada que hice:

> La apertura es la primera parte del juego, aproximadamente los primeros diez movimientos más o menos. En la apertura, es una buena idea hacer tres cosas: saca a relucir tus alfiles y caballos, trata de controlar el centro del tablero y enroca a tu rey.
>
> Por supuesto, estas son solo pautas. Será importante saber cuándo seguir estas pautas y cuándo ignorar estas sugerencias.

Cuando haga clic en **Guardar,** volverá a la página de administración principal para las entradas. Aquí, verá el beneficio de usar text[:50] como la representación de cadena para cada entrada; es mucho más fácil trabajar con varias entradas en la interfaz de administración si solo ve la primera parte de una entrada en lugar del texto completo de cada entrada.

Haga una segunda entrada para Ajedrez y una entrada para Escalada en roca para que tengamos algunos datos iniciales. Aquí hay una segunda entrada para Ajedrez:

> En la fase de apertura del juego, es importante sacar a relucir tus alfiles y caballos. Estas piezas son lo suficientemente poderosas y manejables como para desempeñar un papel importante en los movimientos iniciales de un juego.

Y aquí hay una primera entrada para Rock Climbing:

> Uno de los conceptos más importantes en la escalada es mantener el peso sobre los pies tanto como sea posible. Existe el mito de que los escaladores pueden colgarse todo el día de los brazos. En realidad, los buenos escaladores han practicado formas específicas de mantener su peso sobre los pies siempre que sea posible.

Estas tres entradas nos darán algo con qué trabajar mientras continuamos para desarrollar el Registro de aprendizaje.

La concha de Django

Con algunos datos ingresados, podemos examinar esos datos mediante programación a través de una sesión de terminal interactiva. Este entorno interactivo se denomina *shell de* Django y es un excelente entorno para probar y solucionar problemas de su proyecto. Aquí hay un ejemplo de una sesión de shell interactiva:

```
(ll_env)learning_log$ python manage.py shell
u >>> from learning_logs.models tema de importación
>>> Tema.objetos.todos()
<Conjunto de consultas [<Tema: Ajedrez>, <Tema: Escalada en roca>]>
```

El comando python manage.py shell, ejecutado en un entorno virtual activo, inicia un intérprete de Python que puede usar para explorar los datos almacenados en la base de datos de su proyecto. Aquí, importamos el tema modelo del módulo learning_logs.models u. Luego usamos el método Topic.objects.all()
para obtener todas las instancias del modelo Topic; la lista que se devuelve se denomina conjunto de *consulta.*

Podemos recorrer un conjunto de consultas tal como lo haríamos con una lista. Así es como tú puede ver el ID que se ha asignado a cada objeto de tema:

```
>>> temas = Tema.objetos.todos()
>>> para tema en temas:
... imprimir (tema.id, tema)
```

```
...
1 ajedrez
2 escalada en roca
```

Almacenamos el conjunto de consultas en temas y luego imprimimos el atributo de identificación de cada tema y la representación de cadena de cada tema. Podemos ver que el ajedrez tiene una identificación de 1 y la escalada en roca tiene una identificación de 2.

Si conoce la ID de un objeto en particular, puede usar el método Tema .objects.get() para recuperar ese objeto y examinar cualquier atributo que tenga. Veamos el texto y los valores de fecha agregada para Ajedrez:

```
>>> t = Tema.objetos.get(id=1)
>>> t.texto
'Ajedrez'
>>> t.fecha_añadida
fechahora.fechahora(2019, 2, 19, 1, 55, 31, 98500, tzinfo=<UTC>)
```

También podemos mirar las entradas relacionadas con un tema determinado. Anteriormente definimos el atributo de tema para el modelo de entrada . Era una ForeignKey, una conexión entre cada entrada y un tema. Django puede usar esta conexión para obtener cada entrada relacionada con un tema determinado, como este:

```
u >>> t.entry_set.all()
<QuerySet [<Entrada: La apertura es la primera parte del juego, más o menos...>, <Entrada:

En la fase de apertura del juego, es importante t...>]>
```

Para obtener datos a través de una relación de clave externa, utilice el nombre en minúsculas del modelo relacionado seguido de un guión bajo y la palabra conjunto u.
Por ejemplo, suponga que tiene los modelos Pizza y Topping, y Topping está relacionado con Pizza a través de una clave externa. Si su objeto se llama my_pizza y representa una sola pizza, puede obtener todos los ingredientes de la pizza usando el código my_pizza .topping_set.all().

Usaremos este tipo de sintaxis cuando comencemos a codificar las páginas que los usuarios pueden solicitar. El shell es muy útil para asegurarse de que su código recupere los datos que desea. Si su código funciona como espera en el shell, puede esperar que funcione correctamente en los archivos dentro de su proyecto. Si su código genera errores o no recupera los datos que esperaba, es mucho más fácil solucionar los problemas de su código en el entorno de shell simple que dentro de los archivos que generan páginas web. No nos referiremos mucho al shell, pero debe continuar usándolo para practicar el trabajo con la sintaxis de Django para acceder a los datos almacenados en el proyecto.

Cada vez que modifique sus modelos, deberá reiniciar el shell para ver los efectos de esos cambios. Para salir de una sesión de shell, presione ctrl-D; en Windows, presione ctrl-Z y luego presione enter.

Inténtalo tú mismo

18-2. Entradas cortas: el método __str__() en el modelo Entry actualmente agrega puntos suspensivos a cada instancia de Entry cuando Django lo muestra en el sitio de administración o en el shell. Agregue una instrucción if al método __str__() que agregue puntos suspensivos solo si la entrada tiene más de 50 caracteres. Use el sitio de administración para agregar una entrada que tenga menos de 50 caracteres y verifique que no tenga puntos suspensivos cuando se visualice.

18-3. La API de Django: cuando escribe código para acceder a los datos de su proyecto, está escribiendo una consulta. Lea la documentación para consultar sus datos en https://docs.djangoproject.com/en/2.2/topics/db/queries/. Gran parte de lo que vea le parecerá nuevo, pero le resultará muy útil cuando empiece a trabajar en sus propios proyectos.

18-4. Pizzería: Comience un nuevo proyecto llamado pizzería con una aplicación llamada pizzas. Defina un modelo Pizza con un campo denominado nombre, que contendrá valores de nombre, como Hawaiian y Meat Lovers. Defina un modelo llamado Cobertura con campos llamados pizza y nombre. El campo pizza debe ser una clave externa para Pizza, y el nombre debe poder contener valores como piña, tocino canadiense y embutido.

Registre ambos modelos con el sitio de administración y use el sitio para ingresar algunos nombres de pizza y coberturas. Use el shell para explorar los datos que ingresó.

Creación de páginas: la página de inicio del registro de aprendizaje

La creación de páginas web con Django consta de tres etapas: definición de URL, escritura de vistas y escritura de plantillas. Puede hacerlo en cualquier orden, pero en este proyecto siempre comenzaremos definiendo el patrón de URL. Un patrón de URL describe la forma en que se presenta la URL. También le dice a Django qué buscar cuando hace coincidir una solicitud del navegador con la URL de un sitio para que sepa qué página devolver.

Luego, cada URL se asigna a una vista particular: la función de vista recupera y procesa los datos necesarios para esa página. La función de vista a menudo representa la página utilizando una *plantilla,* que contiene la estructura general de la página. Para ver cómo funciona esto, hagamos la página de inicio para el Registro de aprendizaje. Definiremos la URL para la página de inicio, escribiremos su función de vista y crearemos una plantilla simple.

Debido a que todo lo que estamos haciendo es asegurarnos de que el Registro de aprendizaje funcione como se supone, crearemos una página simple por ahora. Es divertido diseñar una aplicación web en funcionamiento cuando está completa; una aplicación que se ve bien pero no funciona bien no tiene sentido. Por ahora, la página de inicio mostrará solo un título y una breve descripción.

Mapeo de una URL

Los usuarios solicitan páginas ingresando URL en un navegador y haciendo clic en enlaces, por lo que debemos decidir qué URL se necesitan. La URL de la página de inicio es la primera: es la URL base que la gente usa para acceder al proyecto. Por el momento, la URL base, *http://localhost:8000/,* devuelve el sitio Django predeterminado que nos permite saber que el proyecto se configuró correctamente. Cambiaremos esto asignando la URL base a la página de inicio de Learning Log.

En la carpeta principal del proyecto *learning_log ,* abra el archivo *urls.py.* Aquí está el código que deberías ver:

urls.py
```
u de django.contrib administrador de importación
  desde la ruta de importación django.urls

v urlpatrones = [
w ruta('admin/', admin.sitio.urls),
  ]
```

Las primeras dos líneas importan un módulo y una función para administrar las URL para el sitio de administración u. El cuerpo del archivo define la variable urlpatterns v. En este archivo *urls.py ,* que representa el proyecto como un todo, los urlpatterns La variable incluye conjuntos de direcciones URL de las aplicaciones del proyecto. El código en w incluye el módulo admin.site.urls, que define todas las URL que se pueden solicitar desde el sitio de administración.

Necesitamos incluir las URL para learning_logs, así que agregue lo siguiente:

```
  desde django.contrib administrador de importación
  desde la ruta de importación de django.urls, incluye

  patrones de URL = [
      ruta('admin/', admin.sitio.urls),
u ruta('', include('registro_de_aprendizaje.urls')),
  ]
```

Agregamos una línea para incluir el módulo learning_logs.urls en u.

El *urls.py* predeterminado está en la carpeta *learning_log ;* ahora necesitamos crear un segundo archivo *urls.py* en la carpeta *learning_logs .* Cree un nuevo archivo de Python y guárdelo como *urls.py* en *learning_logs* e ingrese este código en él:

urls.py
```
u """Define patrones de URL para learning_logs."""

v desde la ruta de importación de django.urls

w de . importar vistas

x app_name = 'registro_de_aprendizaje'
y patrones de URL = [
      # Página de inicio
z ruta('', vistas.índice, nombre='índice'),
  ]
```

Para dejar claro en qué *urls.py* estamos trabajando, agregamos una cadena de documentación al comienzo del archivo u. Luego importamos la función de ruta , que es necesaria cuando se asignan URL a vistas v. También importamos el módulo de vistas w; el punto le dice a Python que importe el módulo *views.py* desde el mismo directorio que el módulo *urls.py* actual . La variable app_name ayuda a Django a distinguir este archivo *urls.py* de archivos con el mismo nombre en otras aplicaciones dentro del proyecto x. La variable urlpatterns en este módulo es una lista de páginas individuales que se pueden solicitar desde la aplicación learning_logs y.

El patrón de URL real es una llamada a la función path() , que toma tres argumentos z. El primer argumento es una cadena que ayuda a Django a enrutar correctamente la solicitud actual. Django recibe la URL solicitada e intenta enrutar la solicitud a una vista. Para ello, busca en todos los patrones de URL que hemos definido para encontrar uno que coincida con la solicitud actual. Django ignora la URL base del proyecto *(http:// localhost:8000/),* por lo que la cadena vacía ('') coincide con la URL base. Cualquier otra URL no coincidirá con este patrón y Django devolverá una página de error si la URL solicitada no coincide con ningún patrón de URL existente.

El segundo argumento en path() z especifica qué función llamar en *views.py.* Cuando una URL solicitada coincide con el patrón que estamos definiendo, Django llama a la función index() desde *views.py* (escribiremos esta función de vista en la siguiente sección). El tercer argumento proporciona el índice de nombres para este patrón de URL para que podamos consultarlo en otras secciones de código. Siempre que queramos proporcionar un enlace a la página de inicio, usaremos este nombre en lugar de escribir una URL.

Escribir una vista

Una función de vista toma información de una solicitud, prepara los datos necesarios para generar una página y luego envía los datos al navegador, a menudo mediante el uso de una plantilla que define cómo se verá la página.

El archivo *views.py* en *learning_logs* se generó automáticamente cuando ejecutamos el comando python manage.py startapp. Esto es lo que hay en *views.py* ahora mismo:

vistas.py

```
desde django.shortcuts importación render

# Crea tus vistas aquí.
```

Actualmente, este archivo solo importa la función render() , que genera la respuesta en función de los datos proporcionados por las vistas. Abra el archivo de vistas y agregue el siguiente código para la página de inicio:

```
desde django.shortcuts importación render

índice def (solicitud):
    """La página de inicio de Learning Log."""
    devolver procesamiento (solicitud, 'learning_logs/index.html')
```

Cuando una solicitud de URL coincide con el patrón que acabamos de definir, Django busca para una función llamada index() en el archivo *views.py* . Django luego pasa el

solicitud de objeto a esta función de vista. En este caso, no necesitamos procesar ningún dato para la página, por lo que el único código en la función es una llamada a render().

La función render() aquí pasa dos argumentos: la solicitud original objeto y una plantilla que puede usar para construir la página. Escribamos esta plantilla.

Escribir una plantilla

La plantilla define cómo debe verse la página y Django completa los datos relevantes cada vez que se solicita la página. Una plantilla le permite acceder a cualquier dato proporcionado por la vista. Debido a que nuestra vista de la página de inicio no proporcionó datos, esta plantilla es bastante simple.

Dentro de la carpeta *learning_logs* , crea una nueva carpeta llamada *templates.* Dentro de la carpeta de *plantillas* , crea otra carpeta llamada *learning_logs.* Esto puede parecer un poco redundante (tenemos una carpeta llamada *learning_logs* dentro de una carpeta llamada *templates* dentro de una carpeta llamada *learning_logs),* pero configura una estructura que Django puede interpretar sin ambigüedades, incluso en el contexto de un proyecto grande que contiene muchas aplicaciones individuales. Dentro de la carpeta interna *learning_logs* , crea un nuevo archivo llamado *index.html.* La ruta al archivo será *learning_log/ learning _logs/ templates/ learning_logs/ index.html.* Ingrese el siguiente código en ese archivo:

índice.html

```
<p>Registro de aprendizaje</p>

<p>El registro de aprendizaje lo ayuda a realizar un seguimiento de su aprendizaje, para cualquier tema que esté
aprendiendo sobre.</p>
```

Este es un archivo muy simple. Si no está familiarizado con HTML, el <p></p> las etiquetas significan párrafos. La etiqueta <p> abre un párrafo y la etiqueta </p> cierra un párrafo. Tenemos dos párrafos: el primero actúa como título y el segundo describe lo que los usuarios pueden hacer con Learning Log.

Ahora, cuando solicite la URL base del proyecto, *http:// localhost:8000/,* debería ver la página que acabamos de crear en lugar de la página predeterminada de Django. Django tomará la URL solicitada y esa URL coincidirá con el patrón ''; luego Django llamará a la función views.index(), que representará la página usando la plantilla contenida en *index.html.* La figura 18-3 muestra la página resultante.

Figura 18-3: La página de inicio de Learning Log

Aunque puede parecer un proceso complicado para crear una página, esta separación entre URL, vistas y plantillas funciona bastante bien.
Le permite pensar en cada aspecto de un proyecto por separado. En proyectos más grandes, permite que las personas que trabajan en el proyecto se concentren en las áreas en las que son más fuertes. Por ejemplo, un especialista en bases de datos puede enfocarse en los modelos, un programador puede enfocarse en el código de vista y un diseñador web puede enfocarse en las plantillas.

N ota *s posible que vea el siguiente mensaje de error:*

```
ModuleNotFoundError: ningún módulo llamado 'learning_logs.urls'
```

Si lo hace, detenga el servidor de desarrollo presionando ctrl-C en la ventana de terminal donde emitió el comandorunserver . Luego vuelva a emitir el comando python manage servidor de ejecución .py. Debería poder ver la página de inicio. Cada vez que se encuentre con un error como este, intente detener y reiniciar el servidor.

Inténtalo tú mismo

18-5. Planificador de comidas: considere una aplicación que ayude a las personas a planificar sus comidas durante la semana. Cree una nueva carpeta llamada meal_planner e inicie un nuevo proyecto de Django dentro de esta carpeta. Luego crea una nueva aplicación llamada Meal_plans. Haga una página de inicio simple para este proyecto.

18-6. Página de inicio de pizzería: agregue una página de inicio al proyecto de pizzería que comenzó en el ejercicio 18-4 (página 394).

Creación de páginas adicionales

Ahora que hemos establecido una rutina para construir una página, podemos comenzar a construir el proyecto de registro de aprendizaje. Crearemos dos páginas que muestren datos: una página que enumere todos los temas y una página que muestre todas las entradas de un tema en particular. Para cada página, especificaremos un patrón de URL, escribiremos una función de vista y escribiremos una plantilla. Pero antes de hacer esto, crearemos una plantilla base de la que puedan heredar todas las plantillas del proyecto.

Herencia de plantilla

Al crear un sitio web, algunos elementos siempre deberán repetirse en cada página. En lugar de escribir estos elementos directamente en cada página, puede escribir una plantilla base que contenga los elementos repetidos y luego hacer que cada página herede de la base. Este enfoque le permite concentrarse en desarrollar los aspectos únicos de cada página y hace que sea mucho más fácil cambiar la apariencia general del proyecto.

La plantilla principal

Crearemos una plantilla llamada *base.html* en el mismo directorio que *index.html.* Este archivo contendrá elementos comunes a todas las páginas; todas las demás plantillas heredarán de *base.html.* El único elemento que queremos repetir en cada página en este momento es el título en la parte superior. Debido a que incluiremos esta plantilla en cada página, hagamos que el título sea un enlace a la página de inicio:

base.html

```
<p>
u <a href="{% url 'learning_logs:index' %}">Registro de aprendizaje</a>
</p>

v {% contenido de bloque %}{% contenido de bloque final %}
```

La primera parte de este archivo crea un párrafo que contiene el nombre del proyecto, que también actúa como un enlace a la página de inicio. Para generar un enlace, usamos una *etiqueta de plantilla,* que se indica mediante llaves y signos de porcentaje {% %}. Una etiqueta de plantilla genera información que se mostrará en una página. Nuestra etiqueta de plantilla {% url 'learning_logs:index' %} genera una URL que coincide con el patrón de URL definido en *learning_logs/ urls.py* con el nombre 'index' u. En este ejemplo, learning_logs es el espacio de *nombres* y el índice es un patrón de URL con un nombre único en ese espacio de nombres. El espacio de nombres proviene del valor que asignamos a app_name en el archivo *learning_logs/ urls.py .*

En una página HTML simple, un enlace está rodeado por la etiqueta de *anclaje* <a>:

```
<a href="link_url"> texto del enlace</a>
```

Hacer que la etiqueta de la plantilla genere la URL para nosotros lo hace mucho más fácil para mantener nuestros enlaces actualizados. Solo necesitamos cambiar el patrón de URL en *urls.py,* y Django insertará automáticamente la URL actualizada la próxima vez que se solicite la página. Cada página de nuestro proyecto heredará de *base.html,* por lo que a partir de ahora, cada página tendrá un enlace a la página de inicio.

En v insertamos un par de etiquetas de bloque . Este bloque, denominado contenido, es un marcador de posición; la plantilla secundaria definirá el tipo de información que va en el bloque de contenido .

Una plantilla secundaria no tiene que definir cada bloque de su principal, por lo que puede reservar espacio en las plantillas principales para tantos bloques como desee; la plantilla secundaria usa solo los que necesita.

En el código de Python, casi siempre usamos cuatro espacios cuando aplicamos sangría. Los archivos de plantilla tienden a tener más niveles de anidamiento que los archivos de Python, por lo que es común usar solo dos espacios para cada nivel de sangría. Solo necesitas asegurarte de ser consistente.

La plantilla infantil

Ahora necesitamos reescribir *index.html* para heredar de *base.html.* Agregue el siguiente código a *index.html:*

índice.html

```
u {% extiende "learning_logs/base.html" %}

v {% contenido del bloque%}
```

```
    <p>El registro de aprendizaje lo ayuda a realizar un seguimiento de su aprendizaje, para cualquier tema sobre
    el que esté aprendiendo.</p>
w {% contenido de bloque final %}
```

Si compara esto con el *index.html original,* puede ver que hemos reemplazado el título del Registro de aprendizaje con el código para heredar de una plantilla principal u. Una plantilla secundaria debe tener una etiqueta {% extends %} en la primera línea para decirle a Django de qué plantilla principal debe heredar. El archivo *base.html*
es parte de learning_logs, por lo que incluimos *learning_logs* en la ruta a la plantilla principal. Esta línea extrae todo lo contenido en la *plantilla base.html* y permite que *index.html* defina lo que va en el espacio reservado por el bloque de contenido .

Definimos el bloque de contenido en v insertando una etiqueta {% bloque %} con el nombre contenido. Todo lo que no heredamos de la plantilla principal va dentro del bloque de contenido . Aquí, ese es el párrafo que describe el proyecto de registro de aprendizaje. En w indicamos que hemos terminado de definir el contenido usando una etiqueta {% endblock content %} . La etiqueta {% endblock %} no requiere un nombre, pero si una plantilla crece para contener varios bloques, puede ser útil saber exactamente qué bloque está terminando.

Puede comenzar a ver el beneficio de la herencia de plantillas: en una plantilla secundaria, solo necesitamos incluir contenido que sea exclusivo de esa página. Esto no solo simplifica cada plantilla, sino que también facilita mucho la modificación del sitio. Para modificar un elemento común a muchas páginas, solo necesita modificar la plantilla principal. Luego, sus cambios se transfieren a cada página que hereda de esa plantilla. En un proyecto que incluye decenas o cientos de páginas, esta estructura puede hacer que mejorar su sitio sea mucho más fácil y rápido.

En un proyecto grande, es común tener una plantilla principal llamada base.html *para todo el sitio y plantillas principales para cada sección principal del sitio. Todas las plantillas de sección heredan de* base.html, *y cada página del sitio hereda de una plantilla de sección. De esta manera, puede modificar fácilmente la apariencia del sitio como un todo, cualquier sección del sitio o cualquier página individual. Esta configuración proporciona una manera muy eficiente de trabajar y lo alienta a actualizar constantemente su sitio a lo largo del tiempo.*

La página de temas

Ahora que tenemos un enfoque eficiente para crear páginas, podemos centrarnos en las siguientes dos páginas: la página de temas generales y la página para mostrar las entradas de un solo tema. La página de temas mostrará todos los temas que los usuarios han creado y es la primera página que implicará trabajar con datos.

El patrón de URL de temas

Primero, definimos la URL para la página de temas. Es común elegir un fragmento de URL simple que refleje el tipo de información presentada en la página.

Usaremos la palabra *temas,* por lo que la URL *http:// localhost:8000/ topics/* devolverá esta página. Así es como modificamos *learning_logs/ urls.py:*

urls.py

```
"""Define patrones de URL para learning_logs."""
--recorte--
patrones de URL = [
    # Página de inicio.
    ruta('', vistas.índice, nombre='índice'),
    # Página que muestra todos los temas.
u ruta('temas/', vistas.temas, nombre='temas'),
]
```

Simplemente agregamos temas/ en el argumento de cadena utilizado para la URL de la página de inicio u. Cuando Django examina una URL solicitada, este patrón coincidirá con cualquier URL que tenga la URL base seguida de *temas.* Puede incluir u omitir una barra inclinada al final, pero no puede haber nada más después de la palabra *temas* o el patrón no coincidirá. Cualquier solicitud con una URL que coincida con este patrón se pasará a la función themes()

en *vistas.py.*

La vista de temas

La función themes() necesita recuperar algunos datos de la base de datos y enviarlos a la plantilla. Esto es lo que necesitamos agregar a *views.py:*

vistas.py

```
desde django.shortcuts importación render

u del tema de importación de .models

índice def (solicitud):
    --recorte--

v def temas (solicitud):
    """Mostrar todos los temas."""
w temas = Tema.objetos.order_by('fecha_añadida')
x contexto = {'temas': temas}
y return render(solicitud, 'learning_logs/topics.html', context)
```

Primero importamos el modelo asociado con los datos que necesitamos. La función topics() necesita un parámetro: el objeto de solicitud que Django recibió del servidor v. En w consultamos la base de datos solicitando los objetos Topic , ordenados por el atributo date_added . Almacenamos el conjunto de consultas resultante en temas.

En x definimos un contexto que enviaremos a la plantilla. Un *contexto* es un diccionario en el que las claves son nombres que usaremos en la plantilla para acceder a los datos y los valores son los datos que necesitamos enviar a la plantilla. En este caso, hay un par clave-valor, que contiene el conjunto de temas que mostraremos en la página. Al crear una página que utiliza datos, pasamos la variable de contexto a render() , así como el objeto de solicitud y la ruta a la plantilla y.

La plantilla de temas

La plantilla para la página de temas recibe el diccionario de contexto , por lo que la plantilla puede usar los datos que proporciona themes() . Crea un archivo llamado *themes.html* en el mismo directorio que *index.html.* Así es como podemos mostrar los temas en la plantilla:

temas.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

  <p>Temas</p>

tu <ul>
v {% para tema en temas %}
        <li> {{tema}} </li>
wx {% vacío %}
        <li>Todavía no se han agregado temas.</li>
y {% endfor%}
z</ul>

{% contenido de bloque final %}
```

Usamos la etiqueta {% extends %} para heredar de *base.html,* al igual que el índice plantilla lo hace, y luego abra un bloque de contenido . El cuerpo de esta página contiene una lista con viñetas de los temas que se han ingresado. En HTML estándar, una lista con viñetas se denomina *lista desordenada* y se indica con las etiquetas . Comenzamos la lista de temas con viñetas en u.

En v tenemos otra etiqueta de plantilla equivalente a un bucle for , que se repite a través de la lista de temas del diccionario de contexto . El código utilizado en las plantillas difiere de Python en algunos aspectos importantes. Python usa sangría para indicar qué líneas de una instrucción for son parte de un bucle. En una plantilla, cada ciclo for necesita una etiqueta explícita {% endfor %} que indique dónde ocurre el final del ciclo. Entonces, en una plantilla, verá bucles escritos así:

```
{% para el artículo en la lista %}
    hacer algo con cada elemento
{% que antes %}
```

Dentro del bucle, queremos convertir cada tema en un elemento de la lista con viñetas. lista. Para imprimir una variable en una plantilla, envuelva el nombre de la variable entre llaves dobles. Las llaves no aparecerán en la página; simplemente le indican a Django que estamos usando una variable de plantilla. Entonces, el código {{ topic }} en w será reemplazado por el valor de topic en cada pasada por el ciclo. La etiqueta HTML indica un *elemento de lista.* Todo lo que se encuentre entre estas etiquetas, dentro de un par de etiquetas , aparecerá como un elemento con viñetas en la lista.

En x usamos la etiqueta de plantilla {% vacío %} , que le dice a Django qué hacer si no hay elementos en la lista. En este caso, imprimimos un mensaje informando al usuario que aún no se han agregado temas. Las últimas dos líneas cierran el bucle for y y luego cierran la lista con viñetas z.

Ahora necesitamos modificar la plantilla base para incluir un enlace a la página superior de ics. Agrega el siguiente código a *base.html:*

base.html

```
    <p>
u <a href="{% url 'learning_logs:index' %}">Registro de aprendizaje</a> -
v <a href="{% url 'learning_logs:topics' %}">Temas</a>
    </p>

    {% contenido de bloque %}{% contenido de bloque final %}
```

Agregamos un guión después del enlace a la página de inicio u, y luego agregamos un enlace a la página de temas usando la etiqueta de plantilla {% url %} nuevamente v. Esta línea le dice a Django que genere un enlace que coincida con el patrón de URL con el nombre ' temas' en *learning_logs/ urls.py.*

Ahora, cuando actualice la página de inicio en su navegador, verá un vínculo *Temas .* Cuando haga clic en el enlace, verá una página similar a la Figura 18-4.

Figura 18-4: La página de temas

Páginas de temas individuales

A continuación, debemos crear una página que pueda centrarse en un solo tema, que muestre el nombre del tema y todas las entradas de ese tema. Definiremos nuevamente un nuevo patrón de URL, escribiremos una vista y crearemos una plantilla. También modificaremos la página de temas para que cada elemento de la lista con viñetas se vincule a su página de tema correspondiente.

El patrón de URL del tema

El patrón de URL para la página del tema es un poco diferente a los patrones de URL anteriores porque usará el atributo id del tema para indicar qué tema se solicitó. Por ejemplo, si el usuario desea ver la página de detalles del tema Ajedrez, donde la identificación es 1, la URL será *http:// localhost:8000/ topics/ 1/.*

Aquí hay un patrón para que coincida con esta URL, que debe colocar en el *aprendizaje _logs/ urls.py:*

urls.py

```
--recorte--
patrones de URL = [
    --snip-- #
    Página de detalles para un solo tema.
    ruta('temas/<int:id_tema>/', vistas.tema, nombre='tema'),
]
```

Examinemos la cadena 'topics/<int:topic_id>/' en este patrón de URL. La primera parte de la cadena le dice a Django que busque direcciones URL que tengan la palabra *temas* después de la dirección URL base. La segunda parte de la cadena, /<int:topic_id>/, coincide con un número entero entre dos barras diagonales y almacena el valor entero en un argumento llamado topic_id.

Cuando Django encuentra una URL que coincide con este patrón, llama a la función de vista topic() con el valor almacenado en topic_id como argumento. Usaremos el valor de topic_id para obtener el tema correcto dentro de la función.

La vista de tema

La función topic() necesita obtener el tema y todas las entradas asociadas de la base de datos, como se muestra aquí:

vistas.py

```
--recorte--
u def tema (solicitud, topic_id):
        """Mostrar un solo tema y todas sus entradas."""
v tema = Tema.objetos.get(id=id_tema)
w entradas = topic.entry_set.order_by('-date_added')
x contexto = {'tema': tema, 'entradas': entradas}
y return render(solicitud, 'learning_logs/topic.html', context)
```

Esta es la primera función de vista que requiere un parámetro que no sea el objeto de solicitud . La función acepta el valor capturado por la expresión /<int:topic_id>/ y lo almacena en topic_id u. En v usamos get() para recuperar el tema, tal como lo hicimos en el shell de Django. En w obtenemos las entradas asociadas con este tema y las ordenamos de acuerdo con la fecha_añadida. El signo menos delante de date_added ordena los resultados en orden inverso, lo que mostrará primero las entradas más recientes. Almacenamos el tema y las entradas en el diccionario de contexto x y enviamos el contexto a la plantilla *topic.html* y.

Las frases de código en vyw se denominan consultas , porque consultan *la base de datos para obtener información específica. Cuando escribe consultas como estas en sus propios proyectos, es útil probarlas primero en el shell de Django. Obtendrá comentarios mucho más rápidos en el shell que escribiendo una vista y una plantilla, y luego verificando los resultados en un navegador.*

La plantilla de tema

La plantilla debe mostrar el nombre del tema y las entradas. También debemos informar al usuario si aún no se han realizado entradas para este tema.

tema.html

```
{% extiende 'learning_logs/base.html' %}

{% contenido del bloque %}

u <p>Tema: {{ tema }}</p>

    <p>Entradas:</p>
v <ul>
w {% para entrada en entradas %}
      <li>
xy        <p>{{ entrada.fecha_añadida|fecha:'M d, YH:i' }}</p>
          <p>{{ entrada.texto|saltos de línea}}</p>
      </li>
z {% vacío %}
      <li>Todavía no hay entradas para este tema.</li>
    {% que antes %}
    </ul>

{% contenido de bloque final %}
```

Extendemos *base.html,* como lo hacemos con todas las páginas del proyecto. A continuación, mostramos el tema que se está mostrando actualmente, que se almacena en la variable de plantilla {{ topic }}. La variable topic está disponible porque está incluida en el diccionario de contexto . Luego comenzamos una lista con viñetas para mostrar cada una de las entradas v y las repasamos como hicimos con los temas anteriores w.

Cada viñeta enumera dos piezas de información: la marca de tiempo y el texto completo de cada entrada. Para la marca de tiempo x, mostramos el valor del atributo date_added. En las plantillas de Django, una línea vertical (|) representa un *filtro de plantilla, una* función que modifica el valor en una variable de plantilla.
La fecha del filtro :'M d, YH:i' muestra marcas de tiempo en el formato *1 de enero de 2018 23:00.*
La siguiente línea muestra el valor completo del texto en lugar de solo los primeros 50 caracteres de la entrada. El filtro saltos de línea y garantiza que las entradas de texto largo incluyan saltos de línea en un formato comprensible para los navegadores en lugar de mostrar un bloque de texto ininterrumpido.
En z usamos la etiqueta de plantilla {% vacío %} para imprimir un mensaje que informa al usuario que no se han realizado entradas.

Enlaces de la página de temas

Antes de mirar la página del tema en un navegador, debemos modificar la plantilla de temas para que cada tema se vincule a la página adecuada. Este es el cambio que debe realizar en *topics.html:*

temas.html

```
--recorte--
    {% por tema en temas %}
      <li>
        <a href="{% url 'learning_logs:topic' topic.id %}">{{ tema }}</a>
      </li>
    {% vacío %}
--recorte--
```

Usamos la etiqueta de plantilla de URL para generar el enlace adecuado, según el patrón de URL en learning_logs con el nombre 'tema'. Este patrón de URL requiere un argumento topic_id , por lo que agregamos el atributo topic.id a la etiqueta de plantilla de URL. Ahora, cada tema de la lista de temas es un enlace a una página de tema, como *http:// localhost:8000/ topics/ 1/.*

Cuando actualice la página de temas y haga clic en un tema, debería ver un página que se parece a la Figura 18-5.

N ota *xiste una diferencia sutil pero importante entre topic.id y topic_id. La expresión topic.id examina un tema y recupera el valor del ID correspondiente. La variable topic_id es una referencia a esa ID en el código. Si encuentra errores al trabajar con ID, asegúrese de usar estas expresiones de la manera adecuada.*

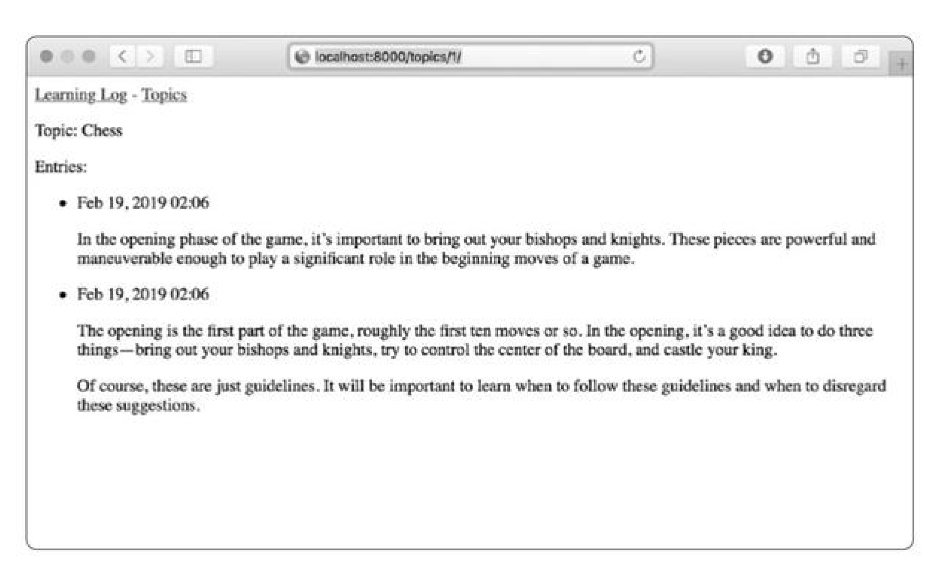


Figura 18-5: La página de detalles de un solo tema, que muestra todas las entradas de un tema

Inténtalo tú mismo

18-7. Documentación de la plantilla: hojee la documentación de la plantilla de Django en https://docs.djangoproject.com/en/2.2/ref/templates/. Puede volver a consultarlo cuando esté trabajando en sus propios proyectos.

18-8. Pizzeria Pages: Agregue una página al proyecto Pizzeria del Ejercicio 18-6 (página 398) que muestre los nombres de las pizzas disponibles. Luego vincule cada nombre de pizza a una página que muestre los ingredientes de la pizza. Asegúrese de utilizar la herencia de plantillas para crear sus páginas de manera eficiente.

Resumen

En este capítulo, aprendió a crear aplicaciones web simples utilizando el marco Django. Escribió una breve especificación del proyecto, instaló Django en un entorno virtual, configuró un proyecto y verificó que el proyecto se configuró correctamente. Configuró una aplicación y definió modelos para representar los datos de su aplicación. Aprendió sobre las bases de datos y cómo Django lo ayuda a migrar su base de datos después de realizar un cambio en sus modelos. Creó un superusuario para el sitio de administración y usó el sitio de administración para ingresar algunos datos iniciales.

También exploró el shell de Django, que le permite trabajar con los datos de su proyecto en una sesión de terminal. Aprendió a definir direcciones URL, crear funciones de visualización y escribir plantillas para crear páginas para su sitio. También utilizó la herencia de plantillas para simplificar la estructura de las plantillas individuales y facilitar la modificación del sitio a medida que evoluciona el proyecto.

En el Capítulo 19, creará páginas intuitivas y fáciles de usar que permitan a los usuarios agregar nuevos temas y entradas y editar entradas existentes sin pasar por el sitio de administración. También agregará un sistema de registro de usuarios, lo que permitirá a los usuarios crear una cuenta y hacer su propio registro de aprendizaje. Este es el corazón de una aplicación web: la capacidad de crear algo con lo que cualquier número de usuarios puede interactuar.



19

Cuentas de usuario

En el corazón de una aplicación web está la posibilidad de que cualquier usuario, en cualquier parte del mundo, registre una cuenta con su aplicación y comience a usarla. En este capítulo, creará formularios para que los usuarios puedan agregar sus propios temas y entradas, y editar las entradas existentes. También aprenderá cómo Django protege contra ataques comunes a páginas basadas en formularios para que no tenga que perder mucho tiempo pensando en proteger sus aplicaciones.

También implementará un sistema de autenticación de usuarios. Creará una página de registro para que los usuarios creen cuentas y luego restringirá el acceso a ciertas páginas solo a los usuarios registrados. Luego, modificará algunas de las funciones de vista para que los usuarios solo puedan ver sus propios datos. Aprenderá a mantener los datos de sus usuarios seguros y protegidos.

Permitir a los usuarios ingresar datos

Antes de construir un sistema de autenticación para crear cuentas, primero agregaremos algunas páginas que permitan a los usuarios ingresar sus propios datos. Daremos a los usuarios la posibilidad de agregar un nuevo tema, agregar una nueva entrada y editar sus entradas anteriores.

Actualmente, solo un superusuario puede ingresar datos a través del sitio de administración. No queremos que los usuarios interactúen con el sitio de administración, por lo que usaremos las herramientas de creación de formularios de Django para crear páginas que permitan a los usuarios ingresar datos.

Agregar nuevos temas

Comencemos por permitir que los usuarios agreguen un nuevo tema. Agregar una página basada en formularios funciona de la misma manera que las páginas que ya hemos creado: definimos una URL, escribimos una función de vista y escribimos una plantilla. La principal diferencia es la adición de un nuevo módulo llamado *formularios.py,* que contendrá los formularios.

El formulario del modelo de tema

Cualquier página que permita a un usuario ingresar y enviar información en una página web es un *formulario,* incluso si no lo parece. Cuando los usuarios ingresan información, debemos *validar* que la información proporcionada es el tipo correcto de datos y no es malicioso, como un código para interrumpir nuestro servidor. Luego necesitamos procesar y guardar información válida en el lugar apropiado en la base de datos. Django auto acopla gran parte de este trabajo.

La forma más sencilla de construir un formulario en Django es usar un *ModelForm,* que utiliza la información de los modelos que definimos en el Capítulo 18 para crear automáticamente un formulario. Escriba su primer formulario en el archivo *forms.py,* que debe crear en el mismo directorio que *models.py:*

formularios.py

```
desde formularios de importación de django

del tema de importación de .models

u class TopicForm(formularios.ModelForm):
        metaclase:
             modelo = Tema
volvo        campos = ['texto']
X            etiquetas = {'texto': ''}
```

Primero importamos el módulo de formularios y el modelo con el que trabajaremos, llamado Tema. En u definimos una clase llamada TopicForm, que hereda de los formularios .ModelForm.

La versión más simple de ModelForm consiste en una clase Meta anidada que le dice a Django en qué modelo basar el formulario y qué campos incluir en el formulario. En v construimos un formulario a partir del modelo Topic e incluimos solo el campo de texto w. El código en x le dice a Django que no genere una etiqueta para el texto campo.

La URL del nuevo_tema

La URL de una nueva página debe ser breve y descriptiva. Cuando el usuario quiera agregar un nuevo tema, lo enviaremos a *http:// localhost:8000/ new_topic/.*
Aquí está el patrón de URL para la página new_topic , que agrega al *aprendizaje _logs/ urls.py:*

urls.py
```
--recorte--
patrones de URL = [
    --recorte--
    # Página para agregar un nuevo tema
    ruta('nuevo_tema/', vistas.nuevo_tema, nombre='nuevo_tema'),
]
```

Este patrón de URL envía solicitudes a la función de vista new_topic(), que escribiremos a continuación.

La función de vista

new_topic() La función new_topic() necesita manejar dos situaciones diferentes: solicitudes iniciales para la página new_topic (en cuyo caso debería mostrar un formulario en blanco) y el procesamiento de cualquier dato enviado en el formulario. Una vez que se procesan los datos de un formulario enviado, es necesario redirigir al usuario a los tem
página:

vistas.py
```
desde django.shortcuts importar renderizar, redirigir

del tema de importación de .models
desde .forms importar TopicForm

--recorte--
def nuevo_tema(solicitud):
    """Agregar un nuevo tema."""
u if solicitud.método != 'POST':
        # No se enviaron datos; crear un formulario en blanco.
en      formulario = TopicForm()
    demás:
        # datos POST enviados; procesar datos.
en      formulario = TopicForm(datos=solicitud.POST)
xy      si formulario.es_válido():
            formulario.guardar()
con         volver redirigir('registro_de_aprendizaje:temas')

    # Mostrar un formulario en blanco o inválido.
{ contexto = {'formulario': formulario}
    return render (solicitud, 'learning_logs/new_topic.html', context)
```

Importamos la redirección de función, que usaremos para redirigir al usuario a la página de temas después de que envíe su tema. La función redirect() toma el nombre de una vista y redirige al usuario a esa vista. También importamos el formulario que acabamos de escribir, TopicForm.

Solicitudes GET y POST

Los dos tipos principales de solicitud que usará al crear aplicaciones web son las solicitudes GET y las solicitudes POST. Utiliza solicitudes *GET* para páginas que solo leen datos del servidor. Por lo general, utiliza solicitudes *POST* cuando el usuario necesita enviar información a través de un formulario. Especificaremos el método POST para procesar todos nuestros formularios. (Existen algunos otros tipos de solicitudes, pero no las usaremos en este proyecto).

La función new_topic() toma el objeto de solicitud como un parámetro. Cuando el usuario solicita inicialmente esta página, su navegador enviará una solicitud GET. Una vez que el usuario haya completado y enviado el formulario, su navegador enviará una solicitud POST. Dependiendo de la solicitud, sabremos si el usuario está solicitando un formulario en blanco (una solicitud GET) o si nos pide que procesemos un formulario completo (una solicitud POST).

La prueba en u determina si el método de solicitud es GET o POST. Si el método de solicitud no es POST, la solicitud probablemente sea GET, por lo que debemos devolver un formulario en blanco (si es otro tipo de solicitud, aún es seguro devolver un formulario en blanco). Creamos una instancia de TopicForm v, la asignamos al formulario variable y enviamos el formulario a la plantilla en el diccionario de contexto {. Debido a que no incluimos argumentos al instanciar TopicForm, Django crea un formulario en blanco que el usuario puede completar.

Si el método de solicitud es POST, el bloque else ejecuta y procesa los datos enviados en el formulario. Creamos una instancia de TopicForm w y le pasamos los datos ingresados por el usuario, almacenados en request.POST. El objeto de formulario que se devuelve contiene la información enviada por el usuario.

No podemos guardar la información enviada en la base de datos hasta que hayamos comprobado que es válida x. El método is_valid() verifica que se hayan completado todos los campos obligatorios (todos los campos de un formulario son obligatorios de forma predeterminada) y que los datos ingresados coinciden con los tipos de campo esperados; por ejemplo, que la longitud del texto es inferior a 200 caracteres, como especificamos en *models.py* en el Capítulo 18. Esta validación automática nos ahorra mucho trabajo. Si todo es válido, podemos llamar a save() y, que escribe los datos del formulario en la base de datos.

Una vez que hayamos guardado los datos, podemos salir de esta página. Usamos redirect() para redirigir el navegador del usuario a la página de temas , donde el usuario debería ver el tema que acaba de ingresar en la lista de temas.

La variable de contexto se define al final de la función de vista y la página se representa mediante la plantilla *new_topic.html,* que crearemos a continuación. Este código se coloca fuera de cualquier bloque if ; se ejecutará si se creó un formulario en blanco y se ejecutará si se determina que un formulario enviado no es válido. Un formulario no válido incluirá algunos mensajes de error predeterminados para ayudar al usuario a enviar datos aceptables.

La plantilla new_topic

Ahora crearemos una nueva plantilla llamada *new_topic.html* para mostrar el formulario que acabamos de crear.

nuevo_tema.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}
  <p>Agregar un nuevo tema:</p>

u <form action="{% url 'learning_logs:new_topic' %}" method='post'>
v {% csrf_token%}
w {{ formulario.as_p }}
x <button name="enviar">Añadir tema</button>
  </formulario>

{% contenido de bloque final %}
```

Esta plantilla amplía *base.html,* por lo que tiene la misma estructura base que el resto de las páginas del Registro de aprendizaje. En u definimos un formulario HTML. El argumento de acción le dice al navegador dónde enviar los datos enviados en el formulario; en este caso, lo enviamos de vuelta a la función de vista new_topic(). El argumento del método le dice al navegador que envíe los datos como una solicitud POST.

Django usa la etiqueta de plantilla {% csrf_token %} v para evitar que los atacantes usen el formulario para obtener acceso no autorizado al servidor (este tipo de ataque se denomina *falsificación de solicitud entre sitios).* En w mostramos el formulario; aquí puedes ver lo simple que Django puede hacer ciertas tareas, como mostrar un formulario. Solo necesitamos incluir la variable de plantilla {{ form.as_p }} para que Django cree todos los campos necesarios para mostrar el formulario automáticamente. El modificador as_p le dice a Django que represente todos los elementos del formulario en formato de párrafo, como una forma sencilla de mostrar el formulario de forma ordenada.

Django no crea un botón de envío para formularios, así que definimos uno en x.

Enlace a la página new_topic

A continuación, incluimos un enlace a la página new_topic en la página de temas :

temas.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

  <p>Temas</p>

  <ul>
    --recorte--
  </ul>

  <a href="{% url 'learning_logs:new_topic' %}">Agregar un nuevo tema</a>

{% contenido de bloque final %}
```

Coloque el enlace después de la lista de temas existentes. La Figura 19-1 muestra el forma resultante. Use el formulario para agregar algunos temas nuevos propios.

Figura 19-1: La página para agregar un nuevo tema

Adición de nuevas entradas

Ahora que el usuario puede agregar un nuevo tema, también querrá agregar nuevas entradas. Definiremos nuevamente una URL, escribiremos una función de vista y una plantilla, y vincularemos a la página. Pero primero, agregaremos otra clase a *forms.py.*

El formulario de modelo de entrada

Necesitamos crear un formulario asociado con el modelo Entry pero esta vez con un poco más de personalización que TopicForm:

formularios.py

```
desde formularios de importación de django

de .modelos importar tema, entrada

clase TopicForm(formularios.ModelForm):
        --recorte--

clase EntryForm(formularios.ModelForm):
        metaclase:
                modelo = Entrada
                campos = ['texto']
                etiquetas = {'texto': 'Entrada:'}
ultravioleta    widgets = {'texto': formularios.Textarea(attrs={'cols': 80})}
```

Actualizamos la instrucción de importación para incluir Entrada y Tema. Creamos una nueva clase llamada EntryForm que hereda de forms.ModelForm. La clase EntryForm tiene una clase Meta anidada que enumera el modelo en el que se basa y el campo que se incluirá en el formulario. Nuevamente le damos al campo 'texto' una etiqueta en blanco u.

En v incluimos el atributo widgets . Un *widget* es un elemento de formulario HTML, como un cuadro de texto de una sola línea, un área de texto de varias líneas o una lista desplegable. Al incluir el atributo de widgets , puede anular las opciones de widgets predeterminadas de Django. Al decirle a Django que use un elemento Forms.Textarea , estamos personalizando

el widget de entrada para el campo 'texto', por lo que el área de texto tendrá 80 columnas de ancho en lugar de las 40 predeterminadas. Esto brinda a los usuarios suficiente espacio para escribir una entrada significativa.

La nueva URL de entrada

Las nuevas entradas deben estar asociadas con un tema en particular, por lo que debemos incluir un argumento topic_id en la URL para agregar una nueva entrada. Aquí está la URL, que agrega a *learning_logs/ urls.py:*

urls.py

```
--recorte--
patrones de URL = [
        --recorte--
        # Página para agregar una nueva entrada
        ruta('nueva_entrada/<int:topic_id>/', vistas.nueva_entrada, nombre='nueva_entrada'),
]
```

Este patrón de URL coincide con cualquier URL con el formulario http://localhost: 8000/nueva_entrada/id/, donde *id* es un número que coincide con el ID del tema. El código <int:topic_id> captura un valor numérico y lo asigna a la variable topic_id. Cuando se solicita una URL que coincida con este patrón, Django envía la solicitud y el ID del tema a la función de vista new_entry() .

La función de vista new_entry()

La función de vista para new_entry es muy parecida a la función para agregar un nuevo tema. Agrega el siguiente código a tu archivo *views.py :*

vistas.py

```
desde django.shortcuts importar renderizar, redirigir

del tema de importación de .models
desde .forms importar TopicForm, EntryForm

--recorte--
def nueva_entrada(solicitud, topic_id):
        """Agregar una nueva entrada para un tema en particular."""
u tema = Tema.objetos.get(id=topic_id)

v if solicitud.método != 'POST':
                # No se enviaron datos; crear un formulario en blanco.
en              formulario = FormularioEntrada()
        demás:
                # datos POST enviados; procesar datos.
X               form = EntryForm(data=request.POST)
                si formulario.es_válido():
yz                      new_entry = form.save(commit=False)
                        new_entry.topic = tema
                        nueva_entrada.guardar()
{                       volver a dirigir ('registro_de_aprendizaje:tema', topic_id=topic_id)

        # Mostrar un formulario en blanco o inválido.
        contexto = {'tema': tema, 'formulario': formulario}
        volver render (solicitud, 'learning_logs/new_entry.html', contexto)
```

Actualizamos la declaración de importación para incluir el formulario de entrada que acabamos de hacer. La definición de new_entry() tiene un parámetro topic_id para almacenar el valor que recibe de la URL. Necesitaremos el tema para representar la página y procesar los datos del formulario, por lo que usamos topic_id para obtener el objeto de tema correcto en usted.

En v comprobamos si el método de solicitud es POST o GET. El bloque if se ejecuta si es una solicitud GET y creamos una instancia en blanco de EntryForm w.

Si el método de solicitud es POST, procesamos los datos creando una instancia de EntryForm, rellenada con los datos POST de la solicitud . objeto x. Luego verificamos si el formulario es válido. Si es así, debemos establecer el atributo de tema del objeto de entrada antes de guardarlo en la base de datos. Cuando llamamos a save(), incluimos el argumento commit=False y para decirle a Django que cree un nuevo objeto de entrada y lo asigne a new_entry sin guardarlo todavía en la base de datos. Establecemos el atributo de tema de nueva_entrada en el tema que extrajimos de la base de datos al comienzo de la función z. Luego llamamos a save() sin argumentos, guardando la entrada en la base de datos con el tema asociado correcto.

La llamada redirect() en { requiere dos argumentos: el nombre de la vista a la que queremos redirigir y el argumento que requiere la función de vista. Aquí, estamos redirigiendo a topic(), que necesita el argumento topic_id. Luego, esta vista representa la página de tema para la que el usuario hizo una entrada, y debería ver su nueva entrada en la lista de entradas.

Al final de la función, creamos un diccionario de contexto y renderizamos el página utilizando la plantilla *new_entry.html* . Este código se ejecutará para un formulario en blanco o para un formulario enviado que se evalúe como no válido.

La plantilla new_entry

Como puede ver en el siguiente código, la plantilla para new_entry es similar a la plantilla para new_topic:

nueva_entrada.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

u <p><a href="{% url 'learning_logs:tema' topic.id %}">{{ tema }}</a></p>

    <p>Añadir una nueva entrada:</p>
v <form action="{% url 'learning_logs:new_entry' topic.id %}" method='post'>
        {% csrf_token%}
        {{ formulario.as_p }}
        <button name='submit'>Añadir entrada</button>
    </formulario>

{% contenido de bloque final %}
```

Mostramos el tema en la parte superior de la página u, para que el usuario pueda ver a qué tema está agregando una entrada. El tema también actúa como un enlace a la página principal de ese tema.

El argumento de acción del formulario incluye el valor topic_id en la URL, por lo que la función de vista puede asociar la nueva entrada con el tema correcto v. Aparte de eso, esta plantilla se parece a *new_topic.html.*

Vinculación a la página new_entry

A continuación, debemos incluir un enlace a la página new_entry de cada página de tema en la plantilla de tema:

tema.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

  <p>Tema: {{ tema }}</p>

  <p>Entradas:</p>
  <p>
    <a href="{% url 'learning_logs:new_entry' topic.id %}">Agregar nueva entrada</a>
  </p>

  <ul>
  --recorte-
  </ul>

{% contenido de bloque final %}
```

Colocamos el enlace para agregar entradas justo antes de mostrar las entradas, porque agregar una nueva entrada será la acción más común en esta página. La figura 19-2 muestra la página new_entry . Ahora los usuarios pueden agregar nuevos temas y tantas entradas como deseen para cada tema. Pruebe la página new_entry agregando algunas entradas a algunos de los temas que ha creado.

Figura 19-2: La página *new_entry*

Edición de entradas

Ahora crearemos una página para que los usuarios puedan editar las entradas que agregaron.

La URL edit_entry

La URL de la página debe pasar el ID de la entrada que se va a editar. Aquí está *learning_logs/ urls.py:*

urls.py

```
--recorte--
patrones de URL = [
    --recorte--
    # Página para editar una entrada.
    ruta('edit_entry/<int:entry_id>/', views.edit_entry, name='edit_entry'),
]
```

El ID pasado en la URL (por ejemplo, *http:// localhost:8000/ editar _entry/ 1/)* se almacena en el parámetro entry_id. El patrón de URL envía solicitudes que coinciden con este formato a la función de visualización edit_entry().

La función de visualización

edit_entry() Cuando la página edit_entry recibe una solicitud GET, la función edit_entry() La función devuelve un formulario para editar la entrada. Cuando la página recibe una solicitud POST con texto de entrada revisado, guarda el texto modificado en la base de datos:

vistas.py

```
desde django.shortcuts importar renderizar, redirigir

de .modelos importar tema, entrada
desde .forms importar TopicForm, EntryForm
--recorte--

def editar_entrada(solicitud, id_entrada):
    """Editar una entrada existente."""
u entrada = Entrada.objetos.get(id=entry_id)
    tema = entrada.tema

    if solicitud.método != 'POST':
        # Solicitud inicial; prellene el formulario con la entrada actual.
en      form = EntryForm(instancia=entrada)
    demás:
        # datos POST enviados; procesar datos.
en      form = EntryForm(instancia=entrada, datos=solicitud.POST)
        si formulario.es_válido():
xy          formulario.guardar()
            volver redirigir('registro_de_aprendizaje:tema', topic_id=tema.id)

    contexto = {'entrada': entrada, 'tema': tema, 'formulario': formulario}
    volver render (solicitud, 'learning_logs/edit_entry.html', contexto)
```

Primero importamos el modelo Entry . En u obtenemos el objeto de entrada que el usuario quiere editar y el tema asociado con esta entrada. En el bloque if , que se ejecuta para una solicitud GET, creamos una instancia de EntryForm con el argumento instancia=entrada v. Este argumento le dice a Django que cree el formulario prellenado con información del objeto de entrada existente. El usuario verá sus datos existentes y podrá editarlos.

Al procesar una solicitud POST, pasamos el argumento instancia=entrada y el argumento datos=solicitud.POST w. Estos argumentos le dicen a Django que cree una instancia de formulario basada en la información asociada con el objeto de entrada existente, actualizada con cualquier dato relevante de request.POST. Luego verificamos si el formulario es válido; si es así, llamamos a save() sin argumentos porque la entrada ya está asociada con el tema correcto x. Luego redirigimos a la página del tema , donde el usuario debería ver la versión actualizada de la entrada que editó.

Si mostramos un formulario inicial para editar la entrada o si el formulario enviado no es válido, creamos el diccionario de contexto y representamos la página usando la plantilla *edit_entry.html .*

La plantilla edit_entry

A continuación, creamos una plantilla *edit_entry.html ,* que es similar a *new_entry.html:*

editar_entrada.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

  <p><a href="{% url 'learning_logs:topic' topic.id %}">{{ tema }}</a></p>

  <p>Editar entrada:</p>

u <form action="{% url 'learning_logs:edit_entry' entry.id %}" method='post'>
      {% csrf_token%}
      {{ formulario.as_p }}
v <button name="enviar">Guardar cambios</button>
  </formulario>

{% contenido de bloque final %}
```

En u, el argumento de acción devuelve el formulario a la función edit_entry() para su procesamiento. Incluimos el ID de entrada como argumento en {% url %} etiqueta, por lo que la función de visualización puede modificar el objeto de entrada correcto. Etiquetamos el botón Enviar como *Guardar cambios* para recordarle al usuario que está guardando ediciones, no creando una nueva entrada v.

Vinculación a la página edit_entry

Ahora necesitamos incluir un enlace a la página edit_entry para cada entrada en la página del tema:

tema.html

```
--recorte--
  {% para entrada en entradas %}
    <li>
      <p>{{ entrada.fecha_añadida|fecha:'M d, YH:i' }}</p>
      <p>{{ entrada.texto|saltos de línea}}</p>
      <p>
        <a href="{% url 'learning_logs:edit_entry' entry.id %}">Editar entrada</a>
      </p>
    </li>
--recorte--
```

Incluimos el enlace de edición después de que se haya mostrado la fecha y el texto de cada entrada. Usamos la etiqueta de plantilla {% url %} para determinar la URL del patrón de URL con nombre edit_entry, junto con el atributo ID de la entrada actual en el ciclo (entry.id). El texto del vínculo *Editar entrada* aparece después de cada entrada en la página. La Figura 19-3 muestra cómo se ve la página del tema con estos enlaces.

Figura 19-3: Cada entrada ahora tiene un enlace para editar esa entrada.

Learning Log ahora tiene la mayor parte de la funcionalidad que necesita. Los usuarios pueden agregar temas y entradas, y leer cualquier conjunto de entradas que deseen. En la siguiente sección, implementaremos un sistema de registro de usuarios para que cualquiera pueda crear una cuenta con Learning Log y crear su propio conjunto de temas y entradas.

Inténtalo tú mismo

19-1. Blog: Inicie un nuevo proyecto de Django llamado Blog. Crear una aplicación llamada blogs en el proyecto y un modelo llamado BlogPost. El modelo debe tener campos como título, texto y fecha_añadida. Cree un superusuario para el proyecto y use el sitio de administración para hacer un par de publicaciones breves. Cree una página de inicio que muestre todas las publicaciones en orden cronológico.

Crea un formulario para hacer nuevas publicaciones y otro para editar publicaciones existentes. Complete sus formularios para asegurarse de que funcionen.

Configuración de cuentas de usuario

En esta sección, configuraremos un sistema de registro y autorización de usuarios para que las personas puedan registrar una cuenta e iniciar y cerrar sesión. Crearemos una nueva aplicación para contener toda la funcionalidad relacionada con el trabajo con los usuarios. Usaremos el sistema de autenticación de usuario predeterminado incluido con Django para hacer la mayor parte del trabajo posible. También modificaremos ligeramente el modelo de tema para que cada tema pertenezca a un usuario determinado.

La aplicación de los usuarios

Comenzaremos creando una nueva aplicación llamada usuarios, usando el comando startapp :

```
(ll_env)learning_log$ python manage.py startapp usuarios (ll_env)learning_log$ ls

u db.sqlite3 learning_log learning_logs ll_env manage.py usuarios
(ll_env)learning_log$ ls usuarios v __init__.py
admin.py apps.py migraciones models.py tests.py views.py
```

Este comando crea un nuevo directorio llamado *usuarios* u con una estructura idéntico a la aplicación learning_logs v.

Agregar usuarios a settings.py

Necesitamos agregar nuestra nueva aplicación a INSTALLED_APPS en *settings.py,* así:

configuración.py

```
--recorte--
APLICACIONES_INSTALADAS = [
    # Mis aplicaciones
    'registro_de_aprendizaje',
    'usuarios',

    # Aplicaciones de django predeterminadas.
    --recorte--
]
--recorte--
```

Ahora Django incluirá la aplicación de usuarios en el proyecto general.

Incluir las URL de los usuarios

A continuación, debemos modificar la raíz *urls.py* para que incluya las URL que escribiremos para la aplicación de los usuarios :

urls.py

```
desde django.contrib administrador de importación
desde la ruta de importación de django.urls, incluye

patrones de URL = [
    ruta('admin/', admin.sitio.urls),
    ruta('usuarios/', include('usuarios.urls')),
    ruta('', include('registro_de_aprendizaje.urls')),
]
```

Agregamos una línea para incluir el archivo *urls.py* de los usuarios. Esta línea coincidirá cualquier URL que comience con la palabra *usuarios,* como *http:// localhost:8000/ users / acceso/.*

La página de inicio de sesión

Primero implementaremos una página de inicio de sesión. Usaremos la vista de inicio de sesión predeterminada que proporciona Django, por lo que el patrón de URL para esta aplicación se ve un poco diferente. Cree un nuevo archivo *urls.py* en el directorio *learning_log/ users/* y agréguele lo siguiente:

urls.py

```
"""Define patrones de URL para los usuarios"""

desde la ruta de importación de django.urls, incluye

u app_name = 'usuarios'
patrones de URL = [
    # Incluir direcciones URL de autenticación predeterminadas.
v ruta('', include('django.contrib.auth.urls')),
]
```

Importamos la función de ruta y luego importamos la función de inclusión para que podamos incluir algunas URL de autenticación predeterminadas que ha definido Django. Estas URL predeterminadas incluyen patrones de URL con nombre, como 'iniciar sesión' y 'cerrar sesión'. Establecemos la variable app_name en 'usuarios' para que Django pueda distinguir estas URL de las URL que pertenecen a otras aplicaciones u. Incluso las URL predeterminadas proporcionadas por Django, cuando se incluyen en el archivo *urls.py* de la aplicación de los usuarios , serán accesibles a través del espacio de nombres de los usuarios .

El patrón de la página de inicio de sesión coincide con la URL *http:// localhost:8000/ users / login/* v. Cuando Django lee esta URL, la palabra *usuarios* le dice a Django que busque en *users/ urls.py,* y *login* le dice que envíe solicitudes a la vista de inicio de sesión predeterminada de Django .

La plantilla de inicio de sesión

Cuando el usuario solicita la página de inicio de sesión, Django utilizará una función de vista predeterminada, pero aún debemos proporcionar una plantilla para la página. El valor por defecto

Las vistas de autenticación buscan plantillas dentro de una carpeta llamada *registro,* por lo que necesitaremos crear esa carpeta. Dentro del directorio *learning_log/ users/ ,* crea un directorio llamado *templates;* dentro de eso, crea otro directorio llamado *registro.* Aquí está la plantilla *login.html ,* que debe guardar en *learning_log / usuarios/ plantillas/ registro:*

iniciar sesión.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

u {% si formulario.errores%}
        <p>Tu nombre de usuario y contraseña no coinciden. Vuelve a intentarlo.</p>
    {% terminara si %}

v <form method="post" action="{% url 'users:login' %}">
        {% csrf_token%}
w {{ formulario.as_p }}

x <button name="enviar">Iniciar sesión</button>
y <input type="hidden" name="next" value="{% url
            'learning_logs:index' %}" />
    </formulario>

{% contenido de bloque final %}
```

Esta plantilla amplía *base.html* para garantizar que la página de inicio de sesión tenga la misma apariencia que el resto del sitio. Tenga en cuenta que una plantilla en una aplicación puede heredar de una plantilla en otra aplicación.

Si el atributo de errores del formulario está configurado, mostramos un mensaje de error u, informe indicando que la combinación de nombre de usuario y contraseña no coincide con nada almacenado en la base de datos.

Queremos que la vista de inicio de sesión procese el formulario, por lo que establecemos el argumento de acción como la URL de la página de inicio de sesión v. La vista de inicio de sesión envía un formulario a la plantilla, y depende de nosotros mostrar el formulario y agregar un botón de envío X. en y incluimos un elemento de formulario oculto, 'siguiente'; el argumento de valor le dice a Django dónde redirigir al usuario después de que haya iniciado sesión correctamente. En este caso, enviamos al usuario de vuelta a la página de inicio.

Enlace a la página de inicio de sesión

Agreguemos el enlace de inicio de sesión a *base.html* para que aparezca en todas las páginas. No queremos que el enlace se muestre cuando el usuario ya haya iniciado sesión, por lo que lo anidamos dentro de una etiqueta {% if %} :

base.html

```
<p>
    <a href="{% url 'learning_logs:index' %}">Registro de aprendizaje</a> - <a href="{% url
    'learning_logs:topics' %}">Temas</a> - u {% si usuario.está_autenticado %}

v Hola, {{ usuario.nombre de usuario }}.
    {% demás %}
w <a href="{% url 'users:login' %}">Iniciar sesión</a>
    {% terminara si %}
```

```
</p>

{% contenido de bloque %}{% contenido de bloque final %}
```

En el sistema de autenticación de Django, cada plantilla tiene una variable de usuario disponible, que siempre tiene un conjunto de atributos is_authenticated : el atributo es True si el usuario está conectado y False si no lo está. Este atributo le permite mostrar un mensaje a los usuarios autenticados y otro a los usuarios no autenticados.

Aquí mostramos un saludo a los usuarios actualmente registrados u. autenticado los usuarios tienen un conjunto de atributos de nombre de usuario adicional , que usamos para personalizar el saludo y recordarle al usuario que ha iniciado sesión v. En w mostramos un enlace a la página de inicio de sesión para los usuarios que no han sido autenticados.

Uso de la página de inicio de sesión

Ya configuramos una cuenta de usuario, así que iniciemos sesión para ver si la página funciona. Vaya a *http:// localhost:8000/ admin/.* Si todavía está conectado como administrador, busque un enlace de cierre de sesión en el encabezado y haga clic en él.

Cuando haya cerrado la sesión, vaya a *http:// localhost:8000/ users/ login/.* Debería ver una página de inicio de sesión similar a la que se muestra en la Figura 19-4. Ingrese el nombre de usuario y la contraseña que configuró anteriormente, y debería regresar a la página de índice. El encabezado de la página de inicio debe mostrar un saludo personalizado con su nombre de usuario.

Figura 19-4: La página de inicio de sesión

Saliendo de tu cuenta

Ahora debemos proporcionar una forma para que los usuarios cierren sesión. Pondremos un enlace en *la base . .html* que cierra la sesión de los usuarios; cuando hagan clic en este enlace, irán a una página que confirma que se han desconectado.

Agregar un enlace de cierre de sesión a base.html

Agregaremos el enlace para cerrar sesión en *base.html* para que esté disponible en todas las páginas. Lo incluiremos en la parte {% if user.is_authenticated %} para que solo los usuarios que ya hayan iniciado sesión puedan verlo:

base.html

```
--recorte-
  {% si usuario.está_autenticado%}
    Hola, {{ usuario.nombre de usuario }}.
    <a href="{% url 'users:logout' %}">Cerrar sesión</a>
  {% demás %}
    --recorte--
```

El patrón de URL con nombre predeterminado para cerrar sesión es simplemente 'cerrar sesión'.

La página de confirmación de cierre de sesión

Los usuarios querrán saber que cerraron sesión correctamente, por lo que la vista de cierre de sesión predeterminada muestra la página usando la plantilla *loged_out.html,* que crearemos ahora. Aquí hay una página simple que confirma que el usuario ha cerrado la sesión. Guarde este archivo en *templates/ registration,* el mismo lugar donde guardó *login.html:*

log_out.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}
  <p>Se ha cerrado la sesión. ¡Gracias por visitarnos!</p>
{% contenido de bloque final %}
```

No necesitamos nada más en esta página, porque *base.html* proporciona enlaces a la página de inicio y la página de inicio de sesión si el usuario desea volver a cualquiera de las páginas.

La Figura 19-5 muestra la página de cierre de sesión tal como aparece para un usuario que acaba de hacer clic en el enlace *Cerrar sesión .* El estilo es mínimo porque nos estamos enfocando en construir un sitio que funcione correctamente. Cuando el conjunto requerido de funciones funcione, diseñaremos el sitio para que luzca más profesional.

Figura 19-5: La página de cierre de sesión confirma que un usuario ha cerrado sesión correctamente.

La página de registro

A continuación, crearemos una página para que los nuevos usuarios puedan registrarse. Usaremos el UserCreationForm predeterminado de Django pero escribiremos nuestra propia función de vista y plantilla.

La URL de registro

El siguiente código proporciona el patrón de URL para la página de registro, de nuevo en *users/ urls.py:*

urls.py

```
"""Define patrones de URL para los usuarios"""

desde la ruta de importación de django.urls, incluye

desde . importar vistas

app_name = 'usuarios'
patrones de URL = [
    # Incluir direcciones URL de autenticación predeterminadas.
    ruta('', include('django.contrib.auth.urls')),
    # Página de registro.
    ruta('registrar/', vistas.registrar, nombre='registrar'),
]
```

Importamos el módulo de vistas de los usuarios, que necesitamos porque estamos escribiendo nuestra propia vista para la página de registro. El patrón de la página de registro coincide con la URL *http:// localhost:8000/ users/ register/* y envía solicitudes a la función register() que estamos a punto de escribir.

La función de vista register()

La función de vista register() debe mostrar un formulario de registro en blanco cuando se solicita la página de registro por primera vez y luego procesar los formularios de registro completos cuando se envían. Cuando un registro es exitoso, la función también necesita iniciar sesión en el nuevo usuario. Agrega el siguiente código a *users/ views.py:*

vistas.py

```
desde django.shortcuts importar renderizar, redirigir
desde django.contrib.auth importar inicio de sesión
de django.contrib.auth.forms importar UserCreationForm

def registro (solicitud):
    """Registrar un nuevo usuario."""
    if solicitud.método != 'POST':
        # Mostrar formulario de registro en blanco.
en      form = UserCreationForm()
    demás:
        # Procesar formulario completado.
en      form = UserCreationForm(data=request.POST)

        si formulario.es_válido():
wx          nuevo_usuario = formulario.guardar()
            # Inicie sesión como usuario y luego rediríjalo a la página de inicio.
```

```
yz              iniciar sesión (solicitud, nuevo_usuario)
                volver redirigir ('learning_logs: índice')

        # Mostrar un formulario en blanco o inválido.
        contexto = {'formulario': formulario}
        volver render (solicitud, 'registro/registrar.html', contexto)
```

Importamos las funciones render() y redirect() . Luego importamos la función de inicio de sesión () para iniciar la sesión del usuario si su información de registro es correcta. También importamos el UserCreationForm predeterminado. En la función register() , verificamos si estamos respondiendo o no a una solicitud POST. Si no lo somos, creamos una instancia de UserCreationForm sin datos iniciales u.

Si respondemos a una solicitud POST, creamos una instancia de UserCreationForm en función de los datos enviados v. Verificamos que los datos sean válidos, en este caso, que el nombre de usuario tenga los caracteres apropiados, las contraseñas coincidan y el el usuario no está tratando de hacer nada malicioso en su envío.

Si los datos enviados son válidos, llamamos al método save() del formulario para guardar el nombre de usuario y el hash de la contraseña en la base de datos x. El guardar () El método devuelve el objeto de usuario recién creado, que asignamos a new_user. Cuando se guarda la información del usuario, iniciamos sesión llamando a login() función con los objetos request y new_user y, que crea una sesión válida para el nuevo usuario. Finalmente, redirigimos al usuario a la página de inicio z, donde un saludo personalizado en el encabezado le indica que su registro fue exitoso.

Al final de la función renderizamos la página, que será una formulario en blanco o un formulario enviado que no es válido.

La plantilla de registro

Ahora cree una plantilla para la página de registro, que será similar a la página de inicio de sesión. Asegúrese de guardarlo en el mismo directorio que *login.html:*

registro.html

```
{% extiende "learning_logs/base.html" %}

{% contenido del bloque %}

  <form method="post" action="{% url 'usuarios:registrar' %}">
    {% csrf_token%}
    {{ formulario.as_p }}

    <button name="enviar">Registrarse</button>
    <input type="hidden" name="next" value="{% url 'learning_logs:index' %}" />
  </formulario>

{% contenido de bloque final %}
```

Usamos el método as_p nuevamente para que Django muestre todos los campos en el formulario de manera adecuada, incluidos los mensajes de error si el formulario no se completa correctamente.

Enlace a la página de registro

A continuación, agregaremos el código para mostrar el enlace de la página de registro a cualquier usuario que no haya iniciado sesión actualmente:

base.html

```
--recorte--
  {% si usuario.está_autenticado%}
    Hola, {{ usuario.nombre de usuario }}.
    <a href="{% url 'users:logout' %}">Cerrar sesión</a>
  {% demás %}
    <a href="{% url 'usuarios:registrar' %}">Registrarse</a> -
    <a href="{% url 'users:login' %}">Iniciar sesión</a>
  {% terminara si %}
--recorte--
```

Ahora los usuarios que han iniciado sesión ven un saludo personalizado y un enlace de cierre de sesión. Los usuarios que no han iniciado sesión ven un enlace de página de registro y un enlace de inicio de sesión. Pruebe la página de registro creando varias cuentas de usuario con diferentes nombres de usuario

En la siguiente sección, restringiremos algunas de las páginas para que estén disponibles solo para usuarios registrados, y nos aseguraremos de que cada tema pertenezca a un usuario específico.

N ota *istema de registro que hemos configurado permite que cualquiera pueda crear cualquier número de cuentas para el Registro de aprendizaje. Pero algunos sistemas requieren que los usuarios confirmen su identidad mediante el envío de un correo electrónico de confirmación al que el usuario debe responder. Al hacerlo, el sistema genera menos cuentas de spam que el sistema simple que estamos usando aquí. Sin embargo, cuando está aprendiendo a crear aplicaciones, es perfectamente apropiado practicar con un sistema de registro de usuarios simple como el que estamos usando.*

Inténtalo tú mismo

19-2. Cuentas de blog: agregue un sistema de registro y autenticación de usuarios al proyecto de blog que inició en el ejercicio 19-1 (página 421). Asegúrese de que los usuarios registrados vean su nombre de usuario en algún lugar de la pantalla y los usuarios no registrados vean un enlace a la página de registro.

Permitir que los usuarios sean dueños de sus datos

Los usuarios deberían poder ingresar datos exclusivos para ellos, por lo que crearemos un sistema para averiguar qué datos pertenecen a qué usuario. Luego, restringiremos el acceso a ciertas páginas para que los usuarios puedan trabajar solo con sus propios datos.

Modificaremos el modelo de tema para que cada tema pertenezca a un usuario específico. Esto también se encargará de las entradas, porque cada entrada pertenece a un tema específico. Comenzaremos restringiendo el acceso a ciertas páginas.

Restricción de acceso con @login_required

Django facilita la restricción del acceso a ciertas páginas a los usuarios registrados a través del decorador @login_required . Un *decorador* es una directiva colocada justo antes de la definición de una función que Python aplica a la función antes de que se ejecute, para modificar el comportamiento del código de la función. Veamos un ejemplo.

Restricción del acceso a la página de temas

Cada tema será propiedad de un usuario, por lo que solo los usuarios registrados pueden solicitar la página de temas. Agrega el siguiente código a *learning_logs/ views.py:*

vistas.py

```
desde django.shortcuts importar renderizar, redirigir
desde django.contrib.auth.decorators import login_required

de .modelos importar tema, entrada
--recorte--

@Necesario iniciar sesión
def temas (solicitud):
    """Mostrar todos los temas."""
    --recorte--
```

Primero importamos la función login_required() . Aplicamos login_required() como decorador de la función de vista de temas () anteponiendo login_required con el símbolo @ . Como resultado, Python sabe ejecutar el código en login_required() antes del código en temas().

El código en login_required() verifica si un usuario ha iniciado sesión y Django ejecuta el código en themes() solo si lo está. Si el usuario no ha iniciado sesión, se le redirige a la página de inicio de sesión.

Para que esta redirección funcione, debemos modificar *settings.py* para que Django sepa dónde encontrar la página de inicio de sesión. Agregue lo siguiente al final de *settings.py:*

configuración.py

```
--recorte--

# Mi configuración
LOGIN_URL = 'usuarios: iniciar sesión'
```

Ahora, cuando un usuario no autenticado solicita una página protegida por el decorador @login_required , Django enviará al usuario a la URL definida por LOGIN_URL en *settings.py.*

Puede probar esta configuración cerrando sesión en cualquier cuenta de usuario y yendo a la página de inicio. Haga clic en el enlace **Temas ,** que debería redirigirlo a la página de inicio de sesión. Luego inicie sesión en cualquiera de sus cuentas y, desde la página de inicio, haga clic nuevamente en el enlace **Temas .** Debería poder acceder a la página de temas.

Restricción del acceso a lo largo del registro de aprendizaje

Django facilita la restricción del acceso a las páginas, pero debe decidir qué páginas proteger. Lo mejor es pensar primero en qué páginas deben no estar restringidas y luego restringir todas las demás páginas del proyecto. Puede corregir fácilmente la restricción excesiva del acceso y es menos peligroso que dejar las páginas confidenciales sin restricciones.

En el Registro de aprendizaje, mantendremos la página de inicio y la página de registro sin restricciones. Restringiremos el acceso a todas las demás páginas.

Aquí está *learning_logs/ views.py* con los decoradores @login_required aplicados a todas las vistas excepto index():

vistas.py

```
--recorte--
@Necesario iniciar sesión
def temas (solicitud):
    --recorte--

@Necesario iniciar sesión
def tema (solicitud, topic_id):
    --recorte--

@Necesario iniciar sesión
def nuevo_tema(solicitud):
    --recorte--

@Necesario iniciar sesión
def nueva_entrada(solicitud, topic_id):
    --recorte--

@Necesario iniciar sesión
def editar_entrada(solicitud, id_entrada):
    --recorte--
```

Intente acceder a cada una de estas páginas mientras está desconectado: será redirigido a la página de inicio de sesión. Tampoco podrá hacer clic en enlaces a páginas como new_topic. Pero si ingresa la URL *http:// localhost:8000/ new_topic/,* será redirigido a la página de inicio de sesión. Debe restringir el acceso a cualquier URL que sea de acceso público y se relacione con datos privados del usuario.

Conexión de datos a ciertos usuarios

A continuación, debemos conectar los datos con el usuario que los envió. Necesitamos conectar solo los datos más altos en la jerarquía a un usuario, y los datos de nivel inferior seguirán. Por ejemplo, en el Registro de aprendizaje, los temas son el nivel más alto de datos en la aplicación y todas las entradas están conectadas a un tema. Siempre que cada tema pertenezca a un usuario específico, podemos rastrear la propiedad de cada entrada en la base de datos.

Modificaremos el modelo de tema agregando una relación de clave externa a un usuario. Entonces tendremos que migrar la base de datos. Finalmente, modificaremos algunas de las vistas para que solo muestren los datos asociados con el usuario conectado actualmente.

Modificación del modelo de tema

La modificación a *models.py* es solo dos líneas:

modelos.py

```
desde modelos de importación django.db
de django.contrib.auth.models usuario de importación

Tema de clase (modelos.Modelo):
    """Un tema sobre el que el usuario está aprendiendo."""
    texto = modelos.CharField(max_length=200)
    date_added = modelos.DateTimeField(auto_now_add=True)
    propietario = modelos.ForeignKey(Usuario, on_delete=modelos.CASCADE)

    def __str__(uno mismo):
        """Retorna una representación de cadena del modelo."""
        devolver auto.texto

Entrada de clase (modelos.Modelo):
    --recorte--
```

Importamos el modelo de usuario de django.contrib.auth. Luego agregamos un campo de propietario al Tema, que establece una relación de clave externa con el Usuario modelo. Si se elimina un usuario, todos los temas asociados con ese usuario también se eliminarán.

Identificación de usuarios existentes

Cuando migramos la base de datos, Django modificará la base de datos para que pueda almacenar una conexión entre cada tema y un usuario. Para realizar la migración, Django necesita saber qué usuario asociar a cada tema existente. El enfoque más simple es comenzar dando todos los temas existentes a un usuario, por ejemplo, el superusuario. Pero primero necesitamos saber la identificación de ese usuario.

Veamos los ID de todos los usuarios creados hasta ahora. Inicie una sesión de shell de Django y emita los siguientes comandos:

```
(ll_env)learning_log$ python manage.py shell u >>> from
django.contrib.auth.models import User v >>> User.objects.all()

<Conjunto de consultas [<Usuario: ll_admin>, <Usuario: eric>, <Usuario: willie>]>
w >>> para usuario en User.objects.all():
...         print(user.username, user.id)
...
ll_admin 1
eric 2 willie 3

>>>
```

En u importamos el modelo de Usuario a la sesión de shell. Luego observamos todos los usuarios que se han creado hasta ahora v. El resultado muestra tres usuarios: *ll_admin, eric* y *willie.*

En w recorremos la lista de usuarios e imprimimos el nombre de usuario y la identificación de cada usuario. Cuando Django pregunte con qué usuario asociar los temas existentes, usaremos uno de estos valores de ID.

Migración de la base de datos

Ahora que conocemos los ID, podemos migrar la base de datos. Cuando hagamos esto, Python nos pedirá que conectemos el modelo Topic a un propietario en particular temporalmente o que agreguemos un valor predeterminado a nuestro archivo *models.py* para indicarle qué hacer. Elija la opción 1:

```
u (ll_env)learning_log$ python manage.py makemigrations learning_logs v Está intentando agregar
un campo 'propietario' que no acepta valores NULL al tema sin un valor predeterminado; no podemos hacer eso (la base
    de datos necesita algo para llenar las filas existentes).
w Seleccione una corrección:
     1) Proporcione un valor predeterminado único ahora (se establecerá en todas las filas existentes con un
         valor nulo para esta columna)
     2) Salga y déjeme agregar un valor predeterminado en models.py x
Seleccione una opción: 1 y Ingrese el valor predeterminado ahora, como
Python válido Los módulos datetime y django.utils.timezone están disponibles,
    por lo que puede hacer, por ejemplo, zona horaria. ahora

    Escriba 'salir' para salir de este indicador
z >>> 1
    Migraciones para 'learning_logs': learning_logs/
       migrations/0003_topic_owner.py
    - Agregar propietario de campo al tema
    (ll_env)registro_de_aprendizaje$
```

Comenzamos emitiendo el comando makemigrations u. En la salida en v, Django indica que estamos tratando de agregar un campo obligatorio (no anulable) a un modelo (tema) existente sin un valor predeterminado especificado. Django nos da dos opciones en w: podemos proporcionar un valor predeterminado en este momento, o podemos salir y agregar un valor predeterminado en *models.py.* En x hemos elegido la primera opción. Django luego nos pide que ingresemos el valor predeterminado y.

Para asociar todos los temas existentes con el usuario administrador original, *ll_admin, ingresé* la ID de usuario de 1 en z. Puede usar la ID de cualquier usuario que haya creado; no tiene que ser un superusuario. Django luego migra la base de datos utilizando este valor y genera el archivo de migración *0003_topic_owner.py,* que agrega el propietario del campo al modelo de tema .

Ahora podemos ejecutar la migración. Introduzca lo siguiente en un entorno virtual activo:

```
    (ll_env)learning_log$ python manage.py migrar Operaciones a
    realizar:
       Aplicar todas las migraciones: administración, autenticación, tipos de contenido, registros de aprendizaje, sesiones
    Ejecutando migraciones:
u Aplicando learning_logs.0003_topic_owner... OK (ll_env)learning_log$
```

Django aplica la nueva migración y el resultado es OK u.

Podemos verificar que la migración funcionó como se esperaba en la sesión de shell, así:

```
u >>> from learning_logs.models tema de importación
v >>> para tema en Tema.objetos.todos():
  ...       print(tema, tema.propietario)
  ...
  Ajedrez ll_admin
  Escalada en roca ll_admin
  >>>
```

Importamos Tema de learning_logs.models u, y luego recorremos todos los temas existentes, imprimiendo cada tema y el usuario al que pertenece v. Puede ver que cada tema ahora pertenece al usuario *ll_admin.* (Si obtiene un error cuando ejecuta este código, intente salir del shell e iniciar un nuevo shell).

Puede simplemente restablecer la base de datos en lugar de migrar, pero perderá todos los datos existentes. Es una buena práctica aprender a migrar una base de datos manteniendo la integridad de los datos de los usuarios. Si desea comenzar con una base de datos nueva, emita el comando python manage.py flush para reconstruir la estructura de la base de datos. Tendrá que crear un nuevo superusuario y todos sus datos desaparecerán.

Restricción del acceso a los temas a los usuarios apropiados

Actualmente, si ha iniciado sesión, podrá ver todos los temas, sin importar con qué usuario haya iniciado sesión. Cambiaremos eso mostrando a los usuarios solo los temas que les pertenecen.

Realice el siguiente cambio en la función themes () en *views.py:*

vistas.py

```
--recorte--
@Necesario iniciar sesión
def temas (solicitud):
    """Mostrar todos los temas."""
    temas = Tema.objetos.filtro(propietario=solicitud.usuario).order_by('fecha_añadida')
    contexto = {'temas': temas}
    volver render (solicitud, 'learning_logs/topics.html', contexto)
--recorte-
```

Cuando un usuario inicia sesión, el objeto de solicitud tiene un conjunto de atributos request.user que almacena información sobre el usuario. La consulta Tema.objetos .filter(owner=request.user) le dice a Django que recupere solo los objetos Topic de la base de datos cuyo atributo de propietario coincida con el usuario actual. Debido a que no estamos cambiando la forma en que se muestran los temas, no necesitamos cambiar la plantilla para la página de temas en absoluto.

Para ver si esto funciona, inicie sesión como el usuario al que conectó todos los temas existentes y vaya a la página de temas. Deberías ver todos los temas. Ahora cierre sesión y vuelva a iniciar sesión como un usuario diferente. La página de temas no debe incluir ningún tema.

Protección de los temas de un usuario

Todavía no hemos restringido el acceso a las páginas de temas, por lo que cualquier usuario registrado podría probar varias URL, como *http:// localhost:8000/ topics/ 1/,* y recuperar páginas de temas que coincidan.

Inténtalo tú mismo. Mientras esté conectado como el usuario propietario de todos los temas, copie la URL o anote el ID en la URL de un tema, y luego cierre sesión y vuelva a iniciar sesión como un usuario diferente. Introduce la URL de ese tema. Debería poder leer las entradas, aunque haya iniciado sesión como un usuario diferente.

Arreglaremos esto ahora realizando una verificación antes de recuperar el pedido entradas en la función de vista topic() :

vistas.py

```
desde django.shortcuts importar renderizar, redirigir
desde django.contrib.auth.decorators import login_required
u de django.http importar Http404

--recorte--
@Necesario iniciar sesión
def tema (solicitud, topic_id):
    """Mostrar un solo tema y todas sus entradas."""
    tema = Tema.objetos.get(id=topic_id)
    # Asegúrese de que el tema pertenezca al usuario actual.
v if tema.propietario != solicitud.usuario:
        elevar Http404

    entradas = topic.entry_set.order_by('-date_added')
    contexto = {'tema': tema, 'entradas': entradas}
    volver render (solicitud, 'learning_logs/topic.html', contexto)
--recorte--
```

Una respuesta 404 es una respuesta de error estándar que se devuelve cuando un recurso solicitado no existe en un servidor. Aquí importamos el Http404 excepción u, que plantearemos si el usuario solicita un tema que no debería ver. Después de recibir una solicitud de tema, nos aseguramos de que el usuario del tema coincida con el usuario conectado actualmente antes de mostrar la página. Si el usuario actual no posee el tema solicitado, lanzamos la excepción Http404 v, y Django devuelve una página de error 404.

Ahora, si intenta ver las entradas de temas de otro usuario, verá un mensaje de *página no encontrada* de Django. En el Capítulo 20, configuraremos el proyecto para que los usuarios vean una página de error adecuada.

Protegiendo la página edit_entry

Las páginas edit_entry tienen URL en la forma http://localhost:8000/edit_entry */ entry_id/,* donde *entry_id* es un número. Protejamos esta página para que nadie pueda usar la URL para obtener acceso a las entradas de otra persona:

vistas.py

```
--recorte--
@Necesario iniciar sesión
def editar_entrada(solicitud, id_entrada):
    """Editar una entrada existente."""
    entrada = Entrada.objetos.get(id=entry_id)
```

```
    tema = entrada.tema
    if tema.propietario != solicitud.usuario:
        elevar Http404

    if solicitud.método != 'POST':
        --recorte--
```

Recuperamos la entrada y el tema asociado a esta entrada. Luego verificamos si el propietario del tema coincide con el usuario actualmente conectado; si no coinciden, generamos una excepción Http404 .

Asociación de nuevos temas con el usuario actual

Actualmente, nuestra página para agregar nuevos temas está rota porque no asocia nuevos temas con ningún usuario en particular. Si intenta agregar un nuevo tema, verá el mensaje de error IntegrityError junto con la restricción NOT NULL falló: learning_logs_topic.owner_id. Django dice que no puede crear un nuevo tema sin especificar un valor para el campo propietario del tema .

Hay una solución sencilla para este problema, porque tenemos acceso al usuario actual a través del objeto de solicitud . Agregue el siguiente código, que asocia el nuevo tema con el usuario actual:

vistas.py

```
--recorte--
@Necesario iniciar sesión
def nuevo_tema(solicitud):
    """Agregar un nuevo tema."""
    if solicitud.método != 'POST':
        # No se enviaron datos; crear un formulario en blanco.
        formulario = TopicForm()
    demás:
        # datos POST enviados; procesar datos.
        formulario = TopicForm(datos=solicitud.POST)
        si formulario.es_válido():
            new_topic = form.save(commit=False)
            nuevo_tema.propietario = solicitud.usuario
ultravioleta    nuevo_tema.guardar()
            volver redirigir('registro_de_aprendizaje:temas')

    # Mostrar un formulario en blanco o inválido. contexto
    = {'formulario': formulario}
    return render (solicitud, 'learning_logs/new_topic.html', contexto)
--recorte--
```

Cuando llamamos por primera vez a form.save(), pasamos el argumento commit=False porque necesitamos modificar el nuevo tema antes de guardarlo en la base de datos u. Luego establecemos el atributo de propietario del nuevo tema para el usuario actual v. Finalmente, llamamos a save() en la instancia de tema recién definida w. Ahora el tema tiene todos los datos necesarios y se guardará con éxito.

Debería poder agregar tantos temas nuevos como desee para tantos diferentes usuarios como quieras. Cada usuario tendrá acceso solo a sus propios datos, ya sea que esté viendo datos, ingresando datos nuevos o modificando datos antiguos.

Inténtalo tú mismo

19-3. Refactorización: hay dos lugares en views.py donde nos aseguramos de que el usuario asociado con un tema coincida con el usuario conectado actualmente. Coloque el código para esta verificación en una función llamada check_topic_owner() y llame a esta función cuando corresponda.

19-4. Protección de new_entry: actualmente, un usuario puede agregar una nueva entrada al registro de aprendizaje de otro usuario ingresando una URL con la ID de un tema que pertenece a otro usuario. Evite este ataque comprobando que el usuario actual sea el propietario del tema de la entrada antes de guardar la nueva entrada.

19-5. Blog protegido: en su proyecto de blog, asegúrese de que cada publicación de blog esté conectada a un usuario en particular. Asegúrese de que todas las publicaciones sean de acceso público, pero solo los usuarios registrados pueden agregar publicaciones y editar publicaciones existentes. En la vista que permite a los usuarios editar sus publicaciones, asegúrese de que el usuario esté editando su propia publicación antes de procesar el formulario.

Resumen

En este capítulo, aprendió a usar formularios para permitir a los usuarios agregar nuevos temas y entradas, y editar entradas existentes. Luego aprendió a implementar cuentas de usuario. Permitió que los usuarios existentes iniciaran y cerraran sesión, y usaron el UserCreationForm predeterminado de Django para permitir que las personas crearan nuevas cuentas.

Después de crear un sistema simple de autenticación y registro de usuarios, restringió el acceso a usuarios registrados para ciertas páginas usando @login_required decorador. Luego atribuyó datos a usuarios específicos a través de una relación de clave externa. También aprendió a migrar la base de datos cuando la migración requiere que especifique algunos datos predeterminados.

Finalmente, aprendió cómo asegurarse de que un usuario solo pueda ver los datos que le pertenecen modificando las funciones de vista. Recuperó los datos apropiados utilizando el método filter() y comparó al propietario de los datos solicitados con el usuario conectado actualmente.

Puede que no siempre sea inmediatamente obvio qué datos debe poner a disposición y qué datos debe proteger, pero esta habilidad vendrá con la práctica. Las decisiones que tomamos en este capítulo para proteger los datos de nuestros usuarios también ilustran por qué trabajar con otros es una buena idea al crear un proyecto: tener a otra persona revisando su proyecto hace que sea más probable que detecte áreas vulnerables.

Ahora tiene un proyecto en pleno funcionamiento ejecutándose en su máquina local. En el capítulo final, diseñará el registro de aprendizaje para que sea visualmente atractivo e implementará el proyecto en un servidor para que cualquier persona con acceso a Internet pueda registrarse y crear una cuenta.

20

Diseñar e implementar una aplicación

Learning Log ahora es completamente funcional, pero no tiene estilo y solo se ejecuta en su máquina local. En este capítulo, diseñará el estilo profesional ject de una manera simple pero profesional y luego implementarlo en un servidor en vivo para que cualquier persona en el mundo pueda crear una cuenta y usarlo.

Para el estilo, usaremos la biblioteca Bootstrap, una colección de herramientas para diseñar aplicaciones web para que se vean profesionales en todos los dispositivos modernos, desde un gran monitor de pantalla plana hasta un teléfono inteligente. Para hacer esto, usaremos la aplicación django-bootstrap4, que también le permitirá practicar el uso de aplicaciones creadas por otros desarrolladores de Django.

Implementaremos Learning Log utilizando Heroku, un sitio que le permite enviar su proyecto a uno de sus servidores, para que esté disponible para cualquier persona con conexión a Internet. También comenzaremos a usar un sistema de control de versiones llamado Git para realizar un seguimiento de los cambios en el proyecto.

Cuando haya terminado con Learning Log, podrá desarrollar aplicaciones web simples, hacer que se vean bien e implementarlas en un servidor en vivo. También podrá utilizar recursos de aprendizaje más avanzados a medida que desarrolle sus habilidades.

Hemos ignorado deliberadamente el estilo hasta ahora para centrarnos primero en la funcionalidad de Learning Log. Esta es una buena manera de abordar el desarrollo, porque una aplicación solo es útil si funciona. Por supuesto, una vez que funciona, la apariencia es crítica, por lo que la gente querrá usarlo.

En esta sección, presentaré la aplicación django-bootstrap4 y le mostraré cómo integrarlo en un proyecto para que esté listo para la implementación en vivo.

La aplicación django-bootstrap4

Usaremos django-bootstrap4 para integrar Bootstrap en nuestro proyecto. Esta aplicación descarga los archivos Bootstrap requeridos, los coloca en una ubicación adecuada en su proyecto y hace que las directivas de estilo estén disponibles en las plantillas de su proyecto.

Para instalar django-bootstrap4, emita el siguiente comando en un entorno virtual activo:

```
(ll_env)learning_log$ pip instalar django-bootstrap4
--recorte--
Django-bootstrap4-0.0.7 instalado con éxito
```

A continuación, debemos agregar el siguiente código para incluir django-bootstrap4 en INSTALLED_APPS en *settings.py:*

configuración.py

```
--recorte--
APLICACIONES_INSTALADAS = [
    # Mis aplicaciones.
    'registro_de_aprendizaje',
    'usuarios',

    # Aplicaciones de terceros.
    'arranque4',

    # Aplicaciones de django predeterminadas.
    'django.contrib.admin',
    --recorte--
```

Inicie una nueva sección llamada Aplicaciones de *terceros* para aplicaciones creadas por otros desarrolladores y agregue 'bootstrap4' a esta sección. Asegúrese de colocar esta sección después de # Mis aplicaciones pero antes de la sección que contiene las aplicaciones predeterminadas de Django.

Uso de Bootstrap para diseñar el registro de aprendizaje

Bootstrap es una gran colección de herramientas de diseño. También tiene varias plantillas que puede aplicar a su proyecto para crear un estilo general. Es mucho más fácil usar estas plantillas que usar herramientas de estilo individuales. Para ver las plantillas que ofrece Bootstrap, vaya a *https:// getbootstrap.com/,* haga clic en **Ejemplos** y busque la sección *Barras* de navegación . Usaremos la plantilla *estática Navbar* , que proporciona una barra de navegación superior simple y un contenedor para el contenido de la página.

La Figura 20-1 muestra cómo se verá la página de inicio después de aplicar Plantilla de Bootstrap a *base.html* y modifique *index.html* ligeramente.

Figura 20-1: La página de inicio del Registro de aprendizaje usando Bootstrap

Modificando base.html

Necesitamos modificar la plantilla *base.html* para acomodar la plantilla de Bootstrap. Presentaré el nuevo *base.html* en partes.

Definición de los encabezados HTML

El primer cambio que haremos en *base.html* define los encabezados HTML en el archivo, por lo que cada vez que se abre una página de registro de aprendizaje, la barra de título del navegador muestra el nombre del sitio. También agregaremos algunos requisitos para usar Bootstrap en nuestras plantillas. Elimine todo en *base.html* y reemplácelo con el siguiente código:

base.html

```
u {% cargar arranque4%}

v <!doctype html>
w <html lang="es">
x <cabeza>
        <juego de caracteres meta="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1,
            encoger para ajustar = no">
y <title>Registro de aprendizaje</title>

z {% bootstrap_css%}
        {% bootstrap_javascript jquery='completo' %}

{ </cabeza>
```

En u cargamos la colección de etiquetas de plantilla disponibles en django bootstrap4. A continuación, declaramos este archivo como un documento HTML v escrito en inglés w. Un archivo HTML se divide en dos partes principales, el *encabezado* y el cuerpo: el encabezado del archivo comienza en x. El encabezado de un archivo HTML no contiene ningún contenido: simplemente le dice al navegador lo que necesita saber para mostrar la página correctamente. En y incluimos un elemento de título para la página, que se mostrará en la barra de título del navegador cada vez que se abra el Registro de aprendizaje.

En z usamos una de las etiquetas de plantilla personalizadas de django-bootstrap4, que le dice a Django que incluya todos los archivos de estilo de Bootstrap. La etiqueta que sigue habilita todo el comportamiento interactivo que podría usar en una página, como las barras de navegación plegables. En { es la etiqueta de cierre </head> .

Definición de la barra de navegación

El código que define la barra de navegación en la parte superior de la página es bastante largo, porque tiene que funcionar bien en pantallas de teléfonos estrechas y monitores de escritorio anchos. Trabajaremos a través de la barra de navegación en secciones.

Aquí está la primera parte de la barra de navegación:

base.html

```
--recorte--
</cabeza>
tu <cuerpo>

v <nav class="navbar navbar-expand-md navbar-light bg-light mb-4 border">

w <a class="navbar-brand" href="{% url 'learning_logs:index'%}">
        Registro de aprendizaje</a>

x <button class="navbar-toggler" type="button" data-toggle="collapse"
        objetivo de datos="#navbarCollapse" aria-controls="navbarCollapse"
        aria-expanded="falso" aria-label="Alternar navegación">
    <span class="navbar-toggler-icon"></botón>
```

El primer elemento es la etiqueta de apertura <body> u. El *cuerpo* de un archivo HTML contiene el contenido que los usuarios verán en una página. En v hay un elemento <nav> que indica la sección de enlaces de navegación de la página. Todo lo contenido en este elemento está diseñado de acuerdo con las reglas de estilo de Bootstrap definidas por los selectores navbar, navbar-expand-md y el resto que ve aquí. un *seleccionador* determina a qué elementos de una página se aplica una determinada regla de estilo. Los selectores navbar-light y bg-light dan estilo a la barra de navegación con un fondo temático claro. El mb en mb-4 es la abreviatura de *margin-bottom;* este selector asegura que aparezca un pequeño espacio entre la barra de navegación y el resto de la página. El selector de borde proporciona un borde delgado alrededor del fondo claro para resaltarlo un poco del resto de la página.

En w configuramos el nombre del proyecto para que aparezca en el extremo izquierdo de la barra de navegación y lo convertimos en un enlace a la página de inicio; aparecerá en todas las páginas del proyecto. El selector de marca de la barra de navegación diseña este enlace para que se destaque del resto de los enlaces y es una forma de marcar el sitio.

En x, la plantilla define un botón que aparece si la ventana del navegador es demasiado estrecho para mostrar toda la barra de navegación horizontalmente. Cuando el

el usuario hace clic en el botón, los elementos de navegación aparecerán en una lista desplegable. La referencia de colapso hace que la barra de navegación se colapse cuando el usuario reduce la ventana del navegador o cuando el sitio se muestra en dispositivos móviles con pantallas pequeñas.

Aquí está la siguiente sección de código que define la barra de navegación:

base.html

```
            --recorte--
          <span class="navbar-toggler-icon"></botón>
u <div class="collapse navbar-collapse" id="navbarCollapse">
          <clase ul = "navbar-nav mr-auto">
vw <li class="elemento-de-navegación">
              <a class="nav-link" href="{% url 'registro_de_aprendizaje:temas'%}">
                  Temas</a></li>
          </ul>
```

En u abrimos una nueva sección de la barra de navegación. El término *div* es la abreviatura de división; construyes una página web dividiéndola en secciones y definiendo reglas de estilo y comportamiento que se aplican a esa sección. Cualquier regla de estilo o comportamiento que se defina en una etiqueta div de apertura afectará todo lo que vea hasta la siguiente etiqueta div de cierre, que se escribe como </div>. Este es el comienzo de la parte de la barra de navegación que se contraerá en pantallas y ventanas estrechas.

En v definimos un nuevo conjunto de enlaces. Bootstrap define los elementos de navegación como elementos en una lista desordenada con reglas de estilo que hacen que no se parezca en nada a una lista. Cada enlace o elemento que necesite en la barra se puede incluir como un elemento en una de estas listas. Aquí, el único elemento de la lista es nuestro enlace a la página Temas w.

Aquí está la siguiente parte de la barra de navegación:

base.html

```
                  --recorte--
          </ul>
en        <ul class="navbar-navml-auto">
en          {% si usuario.está_autenticado%}
              <li class="elemento de navegación">
en              <span class="navbar-text"}">Hola, {{ usuario.nombre de usuario }}.
              </li>
              <li class="elemento de navegación">
                <a class="nav-link" href="{% url 'users:logout' %}">Cerrar sesión</a>
              </li>
            {% demás %}
              <li class="elemento de navegación">
                <a class="nav-link" href="{% url 'users:register' %}">Registrarse</a>
              </li>
              <li class="elemento de navegación">
                <a class="nav-link" href="{% url 'users:login' %}">Iniciar sesión</a></li>
            {% terminara si %}
          </ul>
x</div>

      </nav>
```

En u comenzamos un nuevo conjunto de enlaces usando otra etiqueta de apertura <ul> . Puede tener tantos grupos de enlaces como necesite en una página. Este será el grupo de enlaces relacionados con el inicio de sesión y el registro que aparece en el lado derecho de la barra de navegación. El selector ml-auto es la abreviatura de *margin-left automatic:* este selector examina los otros elementos en la barra de navegación y elabora un margen izquierdo que empuja este grupo de enlaces hacia el lado derecho de la pantalla.

El bloque if en v es el mismo bloque condicional que usamos anteriormente para mostrar mensajes apropiados a los usuarios dependiendo de si han iniciado sesión o no. El bloque es un poco más largo ahora porque algunas reglas de estilo están dentro de las etiquetas condicionales. En w es un elemento <span> . El *elemento span* da estilo a fragmentos de texto, o elementos de una página, que forman parte de una línea más larga. Mientras que los elementos de división crean su propia división en una página, los elementos de extensión son continuos dentro de una sección más grande. Esto puede resultar confuso al principio, porque muchas páginas tienen elementos div profundamente anidados. Aquí, estamos usando el elemento span para aplicar estilo al texto informativo en la barra de navegación, como el nombre del usuario que inició sesión.

Queremos que esta información se vea diferente a un enlace, para que los usuarios no tengan la tentación de hacer clic en estos elementos.

En x cerramos el elemento div que contiene las partes de la barra de navegación que colapsarán en pantallas estrechas, y al final de esta sección cerramos la barra de navegación en general. Si quisiera agregar más enlaces a la barra de navegación, agregaría otro elemento <li> a cualquiera de los grupos <ul> que hemos definido en la barra de navegación usando directivas de estilo idénticas a las que ha visto aquí. .

Todavía hay un poco más que debemos agregar a *base.html.* Necesitamos definir dos bloques que las páginas individuales pueden usar para colocar el contenido específico de esas páginas.

Definición de la parte principal de la página

El resto de *base.html* contiene la parte principal de la página:

base.html

```
    --recorte--
  </nav>

u <rol principal="clase principal"="contenedor">
v <div class="pb-2 mb-2 borde inferior">
        {% bloque page_header %}{% endblock page_header %}
      </div>
w <div>
        {% contenido de bloque %}{% contenido de bloque final %}
      </div>
  </principal>

</cuerpo>

</html>
```

En u abrimos una etiqueta <main> . El elemento *principal* se utiliza para la parte más significativa del cuerpo de una página. Aquí asignamos el selector de arranque

contenedor, que es una forma sencilla de agrupar elementos en una página. Colocaremos dos elementos div en este contenedor.

El primer elemento div v contiene un bloque page_header . Usaremos este bloque para titular la mayoría de las páginas. Para que esta sección se destaque del resto de la página, colocamos algo de relleno debajo del encabezado. El *relleno* se refiere al espacio entre el contenido de un elemento y su borde. El selector pb-2 es una directiva de arranque que proporciona una cantidad moderada de relleno en la parte inferior del elemento con estilo. Un *margen* es el espacio entre el borde de un elemento y otros elementos de la página. Queremos un borde solo en la parte inferior de la página, por lo que usamos el selector border-bottom, que proporciona un borde delgado en la parte inferior del bloque page_header .

En w definimos un elemento div más, que contiene el contenido del bloque. No aplicamos ningún estilo específico a este bloque, por lo que podemos diseñar el contenido de cualquier página como mejor nos parezca. Terminamos el archivo *base.html* con etiquetas de cierre para los elementos principal, cuerpo y html .

Cuando cargue la página de inicio de Learning Log en un navegador, debería ver una barra de navegación de aspecto profesional que coincida con la que se muestra en la Figura 20-1. Intente cambiar el tamaño de la ventana para que sea muy estrecha; un botón debe reemplazar la barra de navegación. Haga clic en el botón y todos los enlaces deberían aparecer en una lista desplegable.

Diseñar la página de inicio usando un Jumbotron

Para actualizar la página de inicio, usaremos un elemento Bootstrap llamado *jumbotron,* que es un cuadro grande que se destaca del resto de la página y puede contener lo que desee. Por lo general, se usa en las páginas de inicio para contener una breve descripción del proyecto general y una llamada a la acción que invita al espectador a participar.

Aquí está el archivo *index.html* revisado:

índice.html

```
{% extiende "learning_logs/base.html" %}

tu {% bloque page_header%}
v <div clase="jumbotron">
w <h1 class="display-3">Haga un seguimiento de su aprendizaje.</h1>

x <p class="lead">Haga su propio Registro de aprendizaje y mantenga una lista de los
        temas sobre los que estás aprendiendo. Siempre que aprendes algo nuevo
        sobre un tema, haga una entrada que resuma lo que ha aprendido.</p>

y <a class="btn btn-lg btn-primary" href="{% url 'usuarios:registrar' %}"
        role="button">Registrar &raquo;</a>
    </div>
z {% endblock page_header%}
```

En u le decimos a Django que estamos a punto de definir lo que va en la página_ bloque de cabecera . Un jumbotron es solo un elemento div con un conjunto de directivas de estilo aplicadas v. El selector jumbotron aplica este grupo de directivas de estilo de la biblioteca Bootstrap a este elemento.

Dentro del jumbotron hay tres elementos. El primero es un mensaje breve, *Haga un seguimiento de su aprendizaje,* que brinda a los visitantes primerizos una idea de lo que hace el Registro de aprendizaje. La clase h1 es un encabezado de primer nivel, y el selector display-3 agrega una apariencia más delgada y alta a este encabezado en particular w. En x incluimos un mensaje más largo que brinda más información sobre lo que el usuario puede hacer con su registro de aprendizaje.

En lugar de simplemente usar un enlace de texto, creamos un botón en y que invita a los usuarios a registrar su cuenta de Registro de aprendizaje. Este es el mismo enlace que en el encabezado, pero el botón se destaca en la página y le muestra al espectador lo que debe hacer para comenzar a usar el proyecto. Los selectores que ve aquí lo diseñan como un botón grande que representa una llamada a la acción. El código » es una *entidad html* que parece dos corchetes de ángulo recto combinados (>>). En z cerramos el bloque page_header . No estamos agregando más contenido a esta página, por lo que no necesitamos definir el bloque de contenido en esta página.

La página de índice ahora se parece a la Figura 20-1 y es una mejora significativa ment sobre nuestro proyecto sin estilo.

Estilo de la página de inicio de sesión

Hemos refinado la apariencia general de la página de inicio de sesión, pero aún no el formulario de inicio de sesión. Hagamos que el formulario se vea consistente con el resto de la página modificando el archivo *login.html :*

iniciar sesión.html

```
    {% extiende "learning_logs/base.html" %}
u {% cargar arranque4%}

v {% bloque page_header%}
        <h2>Inicia sesión en tu cuenta.</h2>
    {% endblock page_header%}

    {% contenido del bloque %}
w <form method="post" action="{% url 'users:login' %}" class="form">
            {% csrf_token%}
x {% formulario bootstrap_form %}
y {% botones %}
                <button name="enviar" class="btn btn-primary">Iniciar sesión</button>
            {% botones finales %}

            <tipo de entrada="oculto" nombre="siguiente"
                value="{% url 'registro_de_aprendizaje:índice' %}" />
        </formulario>

    {% contenido de bloque final %}
```

En u cargamos las etiquetas de la plantilla bootstrap4 en esta plantilla. en v definimos el bloque page_header , que le dice al usuario para qué es la página. Tenga en cuenta que hemos eliminado el bloque {% if form.errors %} de la plantilla; django-bootstrap4 gestiona los errores de formulario automáticamente.

En w agregamos un atributo class="form" , y luego usamos la etiqueta de plantilla {% bootstrap_form %} cuando mostramos el formulario x; esto reemplaza el formulario {{ .as_p }} etiqueta que estábamos usando en el Capítulo 19. La plantilla {% booststrap_form %}

La etiqueta inserta reglas de estilo Bootstrap en los elementos individuales del formulario a medida que se representa el formulario. En y, abrimos una etiqueta de plantilla bootstrap4 {% botones %}, que agrega estilo Bootstrap a los botones.

La Figura 20-2 muestra el formulario de inicio de sesión ahora. La página es mucho más limpia y tiene un estilo consistente y un propósito claro. Intente iniciar sesión con un nombre de usuario o contraseña incorrectos; verá que incluso los mensajes de error tienen un estilo consistente y se integran bien con el sitio en general.

Figura 20-2: La página de inicio de sesión diseñada con Bootstrap

Dar estilo a la página de temas

Asegurémonos de que las páginas para ver información también tengan el estilo adecuado, comenzando con la página de temas:

temas.html

```
{% extiende "learning_logs/base.html" %}

tu {% bloque page_header%}
    <h1>Temas</h1>
{% endblock page_header%}

{% contenido del bloque %}
    <ul>
        {% por tema en temas %}
en          <li><h3>
                <a href="{% url 'learning_logs:topic' topic.id %}">{{ tema }}</a>
            </h3></li>
        {% vacío %}
            <li><h3>Todavía no se han añadido temas.</h3></li>
        {% que antes %}
    </ul>

w <h3><a href="{% url 'learning_logs:new_topic' %}">Agregar un nuevo tema</a></h3>
{% contenido de bloque final %}
```

No necesitamos la etiqueta {% load bootstrap4 %} , porque no estamos usando ninguna etiquetas de plantilla bootstrap4 personalizadas en este archivo. Movemos el encabezado *Temas* en el bloque page_header y asígnele un estilo de encabezado en lugar de usar la etiqueta de párrafo simple u. Diseñamos cada tema como un elemento <h3> para hacerlos un poco más grandes en la página v y hacemos lo mismo con el enlace para agregar un nuevo tema w.

Dar estilo a las entradas en la página del tema

La página de temas tiene más contenido que la mayoría de las páginas, por lo que necesita un poco más de trabajo. Usaremos el componente de tarjeta de Bootstrap para que cada entrada se destaque. Una *tarjeta* es un div con un conjunto de estilos flexibles y predefinidos que es perfecto para mostrar las entradas de un tema:

tema.html

```
{% extiende 'learning_logs/base.html' %}

tu {% bloque page_header%}
      <h3>{{ tema }}</h3>
   {% endblock page_header%}

   {% contenido del bloque %}
      <p>
         <a href="{% url 'learning_logs:new_entry' topic.id %}">Agregar nueva entrada</a>
      </p>

      {% para entrada en entradas %}
v <div clase="tarjeta mb-3">
w           <h4 class="tarjeta-encabezado">
  {{ entrada.fecha_añadida|fecha:'M d, YH:i' }}
X              <small><a href="{% url 'learning_logs:edit_entry' entry.id %}">
                  editar entrada</a></pequeño>
            </h4>
y           <div class="cuerpo-de-tarjeta">
               {{entrada.texto|saltos de línea}}
            </div>
         </div>
      {% vacío %}
         <p>Todavía no hay entradas para este tema.</p>
      {% que antes %}

   {% contenido de bloque final %}
```

Primero colocamos el tema en el bloque page_header u. Luego eliminamos la estructura de lista desordenada utilizada anteriormente en esta plantilla. En lugar de convertir cada entrada en un elemento de lista, creamos un elemento div con la tarjeta selectora en v.
Esta tarjeta tiene dos elementos anidados: uno para contener la marca de tiempo y el enlace para editar la entrada, y otro para contener el cuerpo de la entrada.

El primer elemento de la tarjeta es un encabezado, que es un elemento <h4> con el selector card-header w. Este encabezado de tarjeta contiene la fecha en que se realizó la entrada y un enlace para editar la entrada. La etiqueta <small> alrededor del enlace edit_entry

hace que parezca un poco más pequeña que la marca de tiempo x. El segundo elemento es un div con el selector card-body y, que coloca el texto de la entrada en un cuadro simple en la tarjeta. Tenga en cuenta que el código de Django para incluir la información en la página no ha cambiado; solo han cambiado los elementos que afectan la apariencia de la página.

La Figura 20-3 muestra la página del tema con su nueva apariencia. La funcionalidad de Learning Log no ha cambiado, pero ahora se ve más profesional y atractivo para los usuarios.

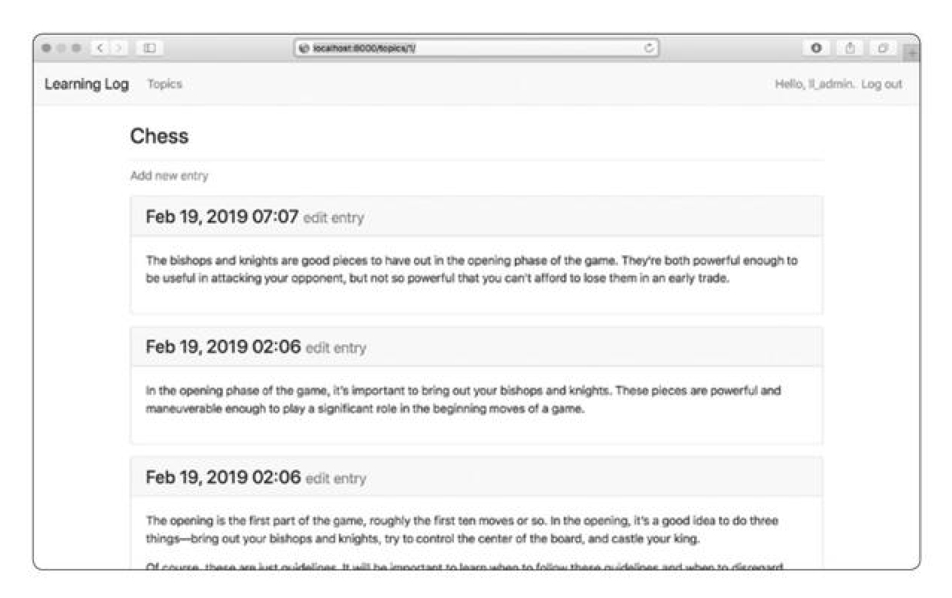
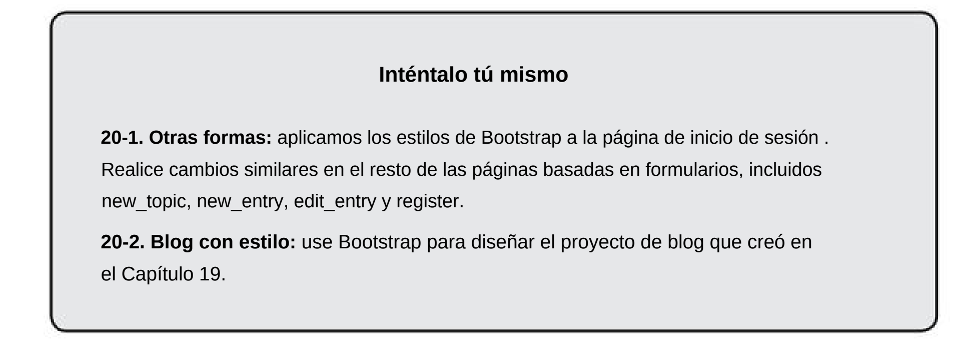


Figura 20-3: La página del tema con estilo Bootstrap

N ota *desea utilizar una plantilla de Bootstrap diferente, siga un proceso similar al que hemos hecho hasta ahora en este capítulo. Copie la plantilla que desea usar en* base.html *y modifique los elementos que contienen contenido real para que la plantilla muestre la información de su proyecto. Luego use las herramientas de estilo individuales de Bootstrap para diseñar el contenido de cada página.*



Inténtalo tú mismo

20-1. Otras formas: aplicamos los estilos de Bootstrap a la página de inicio de sesión . Realice cambios similares en el resto de las páginas basadas en formularios, incluidos new_topic, new_entry, edit_entry y register.

20-2. Blog con estilo: use Bootstrap para diseñar el proyecto de blog que creó en el Capítulo 19.

Implementación del registro de aprendizaje

Ahora que tenemos un proyecto de aspecto profesional, implementémoslo en un servidor en vivo para que cualquier persona con conexión a Internet pueda usarlo. Usaremos Heroku, una plataforma basada en web que le permite administrar la implementación de aplicaciones web. Haremos que Learning Log esté listo y funcionando en Heroku.

Hacer una cuenta de Heroku

Para crear una cuenta, vaya a *https:// heroku.com/* y haga clic en uno de los enlaces de registro. Es gratis crear una cuenta, y Heroku tiene un nivel gratuito que le permite probar sus proyectos en una implementación en vivo antes de implementarlos correctamente.

en cuenta *que el nivel gratuito de Heroku tiene límites, como la cantidad de aplicaciones que puede implementar y cómo a menudo, las personas pueden visitar su aplicación. Pero estos límites son lo suficientemente generosos como para permitirle practicar la implementación de aplicaciones sin costo alguno.*

Instalación de la CLI de Heroku

Para implementar y administrar un proyecto en los servidores de Heroku, necesitará las herramientas disponibles en la interfaz de línea de comandos (CLI) de Heroku. Para instalar la última versión de Heroku CLI, visite *https:// devcenter.heroku.com/ articles/ heroku-cli/* y siga las instrucciones para su sistema operativo. Las instrucciones incluirán un comando de terminal de una línea o un instalador que puede descargar y ejecutar.

Instalación de paquetes necesarios

También deberá instalar tres paquetes que ayuden a servir proyectos de Django en un servidor en vivo. En un entorno virtual activo, emita los siguientes comandos:

```
(ll_env)learning_log$ pip install psycopg2==2.7.*
(ll_env)learning_log$ pip instalar django-heroku
(ll_env)learning_log$ pip instalar gunicorn
```

Se requiere el paquete psycopg2 para administrar la base de datos que usa Heroku. El paquete django-heroku maneja casi toda la configuración que necesita nuestra aplicación para funcionar correctamente en los servidores de Heroku. Esto incluye administrar la base de datos y almacenar archivos estáticos en un lugar donde se puedan servir correctamente. *Los archivos estáticos* contienen reglas de estilo y archivos JavaScript. El paquete gunicorn proporciona un servidor capaz de servir aplicaciones en un entorno en vivo.

Creación de un archivo requirements.txt

Heroku necesita saber de qué paquetes depende nuestro proyecto, por lo que usaremos pip para generar un archivo que los enumere. Nuevamente, desde un entorno virtual activo, emita el siguiente comando:

```
(ll_env)learning_log$ pip congelar > requisitos.txt
```

El comando congelar le dice a pip que escriba los nombres de todos los paquetes actualmente instalados en el proyecto en el archivo *requirements.txt.* Abra este archivo para ver los paquetes y los números de versión instalados en su proyecto:

requisitos.txt

```
dj-base de datos-url==0.5.0
Django==2.2.0
Django-bootstrap4==0.0.7
django-heroku==0.3.1
gunicorn==19.9.0
psicopg2==2.7.7
Pitz == 2018.9
sqlparse==0.2.4
ruido blanco==4.1.2
```

Learning Log ya depende de ocho paquetes diferentes con números de versión específicos, por lo que requiere un entorno específico para funcionar correctamente. (Instalamos cuatro de estos paquetes manualmente y cuatro de ellos se instalaron automáticamente como dependencias de estos paquetes).

Cuando implementamos Learning Log, Heroku instalará todos los paquetes enumerados en *requirements.txt,* creando un entorno con los mismos paquetes que estamos usando localmente. Por esta razón, podemos estar seguros de que el proyecto implementado se comportará igual que en nuestro sistema local. Esta es una gran ventaja a medida que comienza a construir y mantener varios proyectos en su sistema.

Si un paquete aparece en su sistema pero el número de versión difiere del que se muestra aquí, conserve la versión que tiene en su sistema.

Especificación del tiempo de ejecución de Python

A menos que especifique una versión de Python, Heroku utilizará su versión predeterminada actual de Python. Asegurémonos de que Heroku use la misma versión de Python que estamos usando. En un entorno virtual activo, emita el comando python --versión:

```
(ll_env)learning_log$ python --versión
Pitón 3.7.2
```

En este ejemplo, estoy ejecutando Python 3.7.2. Cree un nuevo archivo llamado *runtime.txt* en el mismo directorio que *manage.py* e ingrese lo siguiente:

tiempo de ejecución.txt

```
pitón-3.7.2
```

Este archivo debe contener una línea con su versión de Python especificada en el formato mostrado; asegúrese de ingresar python en minúsculas, seguido de un guión, seguido del número de versión de tres partes.

Si recibe un error que informa que el tiempo de ejecución de Python que solicitó no está disponible, vaya a https://devcenter.heroku.com/categories/language-support/ *y busque un enlace para* especificar un tiempo de ejecución de Python. *Explore el artículo para encontrar los tiempos de ejecución disponibles y use el que más se asemeje a su versión de Python.*

Modificando settings.py para Heroku

Ahora necesitamos agregar una sección al final de *settings.py* para definir algunas configuraciones específicas para el entorno de Heroku:

configuración.py

```
--recorte--
# Mi configuración
LOGIN_URL = 'usuarios: iniciar sesión'

# Configuración de Heroku.
importar django_heroku
django_heroku.settings(locales())
```

Aquí importamos el módulo django_heroku y llamamos a la función settings() . Esta función modifica algunas configuraciones que necesitan valores específicos para el entorno de Heroku.

Hacer un Procfile para Iniciar Procesos

Un *Procfile* le dice a Heroku qué procesos debe comenzar para servir adecuadamente al proyecto. Guarde este archivo de una línea como *Procfile,* con una *P* mayúscula y sin extensión de archivo, en el mismo directorio que *manage.py.*

Aquí está la línea que va en *Procfile:*

perfil

```
web: gunicorn learning_log.wsgi --archivo de registro-
```

Esta línea le dice a Heroku que use gunicorn como servidor y que use el conjunto tings en *learning_log/ wsgi.py* para iniciar la aplicación. El indicador del archivo de registro le dice a Heroku los tipos de eventos que debe registrar.

Uso de Git para rastrear los archivos del proyecto

Como se discutió en el Capítulo 17, Git es un programa de control de versiones que le permite tomar una instantánea del código en su proyecto cada vez que implementa una nueva función con éxito. Si algo sale mal, puede volver fácilmente a la última instantánea de trabajo de su proyecto; por ejemplo, si accidentalmente introduce un error mientras trabaja en una nueva función. Cada instantánea se denomina *confirmación.*

Con Git, puede intentar implementar nuevas funciones sin preocuparse por romper su proyecto. Cuando está implementando en un servidor en vivo, debe asegurarse de que está implementando una versión funcional de su proyecto. Para leer más sobre Git y el control de versiones, consulte el Apéndice D.

Instalando Git

Es posible que Git ya esté instalado en su sistema. Para saber si Git ya está instalado, abra una nueva ventana de terminal y emita el comando git --versión:

```
(ll_env)learning_log$ git --version
git versión 2.17.0
```

Si recibe un mensaje de error por algún motivo, consulte las instrucciones de instalación de Git en el Apéndice D.

Configuración de Git

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando solo una persona está trabajando en el proyecto. Para hacer esto, Git necesita saber su nombre de usuario y correo electrónico. Debe proporcionar su nombre de usuario, pero puede crear un correo electrónico para sus proyectos de práctica:

```
(ll_env)learning_log$ git config --usuario global.nombre "ehmatthes"
(ll_env)learning_log$ git config --usuario global.email "eric@example.com"
```

Si olvida este paso, Git le pedirá esta información cuando realice su primera confirmación.

Ignorar archivos

No necesitamos que Git rastree cada archivo en el proyecto, así que le diremos que ignore algunos archivos. Cree un archivo llamado .gitignore en la carpeta que contiene *manage.py.*
Tenga en cuenta que este nombre de archivo comienza con un punto y no tiene extensión de archivo. Aquí está el código que va en .gitignore:

.gitignore

```
ll_env/
__pycache__/
*.sqlite3
```

Le decimos a Git que ignore todo el directorio *ll_env* , porque podemos volver a crear automáticamente en cualquier momento. Tampoco rastreamos el directorio *__pycache__* , que contiene los *archivos .pyc* que se crean automáticamente cuando Django ejecuta los archivos .py . No hacemos un seguimiento de los cambios en la base de datos local, porque es un mal hábito: si alguna vez usa SQLite en un servidor, puede sobrescribir accidentalmente la base de datos en vivo con su base de datos de prueba local cuando envía el proyecto al servidor. El asterisco en *.sqlite3 le dice a Git que ignore cualquier archivo que termine con la extensión .sqlite3.

Si usa macOS, agregue .DS_Store a su archivo .gitignore . *Este es un archivo que almacena información sobre la configuración de carpetas en macOS y no tiene nada que ver con este proyecto.*

Hacer visibles los archivos ocultos

La mayoría de los sistemas operativos ocultan archivos y carpetas que comienzan con un punto, como .gitignore. Cuando abre un explorador de archivos o intenta abrir un archivo desde una aplicación como Sublime Text, no verá este tipo de archivos de forma predeterminada.
Pero como programador, necesitará verlos. Aquí se explica cómo ver los archivos ocultos, según su sistema operativo:

- En Windows, abra el Explorador de Windows y luego abra una carpeta como *Escritorio.* Haga clic en la pestaña **Ver** y asegúrese de que las **extensiones de nombre de archivo** y los **elementos ocultos** estén marcados.

- En macOS, puede presionar ÿ-shift-. (punto) en cualquier ventana del explorador de archivos para ver los archivos y carpetas ocultos.

- En sistemas Linux como Ubuntu, puede presionar ctrl-H en cualquier explorador de archivos para mostrar archivos y carpetas ocultos. Para que esta configuración sea permanente, abra un explorador de archivos como Nautilus y haga clic en la pestaña de opciones (indicada por tres líneas). Seleccione la casilla de verificación **Mostrar archivos ocultos .**

Comprometer el proyecto

Necesitamos inicializar un repositorio Git para Learning Log, agregar todos los archivos necesarios al repositorio y confirmar el estado inicial del proyecto.
Así es como se hace:

```
u (ll_env)learning_log$ git init
  Repositorio Git vacío inicializado en /home/ehmatthes/pcc/learning_log/.git/
v (ll_env)learning_log$ git agregar.
w (ll_env)learning_log$ git commit -am "Listo para implementación en heroku".
  [master (root-commit) 79fef72] Listo para su implementación en heroku. 45
   archivos cambiados, 712 inserciones (+) modo de creación 100644 .gitignore
   modo de creación 100644 Procfile

   --recorte--
   crear modo 100644 usuarios/vistas.py
x (ll_env)learning_log$ estado de git
  En maestro de rama
  nada que cometer, árbol de trabajo limpio
  (ll_env)registro_de_aprendizaje$
```

En u emitimos el comando git init para inicializar un repositorio vacío en el directorio que contiene el registro de aprendizaje. En v usamos el comando git . con add , que agrega todos los archivos que no se ignoran al repositorio. (No olvide el punto). En w emitimos el comando git commit -am *commit message:* el indicador -a le dice a Git que incluya todos los archivos modificados en este compromiso, y el indicador -m le dice a Git que grabe un mensaje de registro.

Emitir el comando de estado de git x indica que estamos en el *maestro* rama y que nuestro árbol de trabajo esté *limpio.* Este es el estado que querrá ver cada vez que envíe su proyecto a Heroku.

Empujando a Heroku

Finalmente estamos listos para llevar el proyecto a Heroku. En un entorno virtual activo, emita los siguientes comandos:

```
u (ll_env)learning_log$ inicio de sesión heroku
  heroku: Presione cualquier tecla para abrir el navegador para iniciar sesión o q para
  salir: Iniciar sesión... hecho
  Inicia sesión como eric@example.com
v (ll_env)learning_log$ heroku crear
  Creando aplicación... hecho, Ã secret-lowlands-82594
  https://tierras-bajas-secretas-82594.herokuapp.com/ |
      https://git.heroku.com/secret-lowlands-82594.git
```

```
w (ll_env)learning_log$ git push heroku maestro
  --recorte--
  remoto: -----> Iniciando...
  remoto:           Lanzado v5
x remoto: https://secret-lowlands-82594.herokuapp.com/ implementado en Heroku
  remoto: Verificando implementación... hecho.
  A https://git.heroku.com/secret-lowlands-82594.git
   * [nueva rama]          maestro -> maestro
  (ll_env)learning_log$
```

Primero emite el comando de inicio de sesión de heroku , que lo llevará a una página en su navegador donde puede iniciar sesión en su cuenta de Heroku u. Luego le dices a Heroku que construya un proyecto vacío v. Heroku genera un nombre compuesto por dos palabras y un número; puede cambiar esto más adelante. A continuación, emitimos el comando git push heroku master w, que le dice a Git que envíe la rama maestra del proyecto al repositorio que acaba de crear Heroku. Luego, Heroku construye el proyecto en sus servidores utilizando estos archivos. En x está la URL que usaremos para acceder al proyecto en vivo, que podemos cambiar junto con el nombre del proyecto.

Cuando haya emitido estos comandos, el proyecto se implementará pero no se configurará por completo. Para verificar que el proceso del servidor se inició correctamente, use el comando heroku ps :

```
  (ll_env)learning_log$ heroku pd
u Cuota de horas de dinamómetro gratis restante este mes: 450h 44m (81 %)
  Uso gratuito de dinamómetro para esta aplicación: 0h 0m (0%)
  Para obtener más información sobre la suspensión del dinamómetro y cómo actualizar, consulte:
  https://devcenter.heroku.com/articles/dyno-sleeping
v === web (Gratis): gunicorn learning_log.wsgi --log-file - (1)
  web.1: hasta 2019/02/19 23:40:12 -0900 (hace ~ 10m)
  (ll_env)registro_de_aprendizaje$
```

El resultado muestra cuánto tiempo más puede estar activo el proyecto en el próximo mes u. En el momento de escribir este artículo, Heroku permite que las implementaciones gratuitas estén activas hasta 550 horas en un mes. Si un proyecto excede este límite, se mostrará una página de error del servidor estándar; personalizaremos esta página de error en breve. En v vemos que se ha iniciado el proceso definido en *Procfile* .

Ahora podemos abrir la aplicación en un navegador usando el comando heroku open:

```
(ll_env)registro_de_aprendizaje$ heroku abierto
(ll_env)registro_de_aprendizaje$
```

Este comando le evita abrir un navegador e ingresar la URL que le mostró Heroku, pero esa es otra forma de abrir el sitio. Debería ver la página de inicio de Learning Log, con el estilo correcto. Sin embargo, aún no puede usar la aplicación porque no hemos configurado la base de datos.

El proceso de implementación de Heroku cambia de vez en cuando. Si tiene problemas irresolubles, consulte la documentación de Heroku para obtener ayuda. Vaya a https://devcenter.heroku .com/, *haga clic en* ***Python*** *y busque un enlace para* Comenzar con Python *o* Implementar aplicaciones de Python y Django en Heroku. *Si no entiende lo que ve allí, consulte las sugerencias en el Apéndice C.*

Configuración de la base de datos en Heroku

Necesitamos ejecutar la migración una vez para configurar la base de datos en vivo y aplicar todas las migraciones que generamos durante el desarrollo. Puede ejecutar los comandos de Django y Python en un proyecto de Heroku usando el comando heroku run.

Aquí se explica cómo ejecutar la migración en la implementación de Heroku:

```
u (ll_env)learning_log$ heroku ejecutar python manage.py migrate v
Ejecutando 'python manage.py migrate' en Ã secret-lowlands-82594... up, run.3060
    --recorte--
w Ejecutando migraciones:
    --snip--
    Aplicando learning_logs.0001_initial... Aceptar
    Aplicando learning_logs.0002_entry... Aceptar
    Aplicando learning_logs.0003_topic_owner... Aceptar
    Aplicando sesiones.0001_initial... Aceptar
(ll_env)learning_log$
```

Primero emitimos el comando heroku run python manage.py migrate u. Heroku luego crea una sesión de terminal para ejecutar el comando de migración v. En w Django aplica las migraciones predeterminadas y las migraciones que generamos durante el desarrollo de Learning Log.

Ahora, cuando visite su aplicación implementada, debería poder usarla tal como lo hizo en su sistema local. Pero no verá ninguno de los datos que ingresó en su implementación local, incluida su cuenta de superusuario, porque no copiamos los datos en el servidor en vivo. Esta es una práctica normal: no suele copiar datos locales en una implementación en vivo porque los datos locales suelen ser datos de prueba.

Puede compartir su enlace de Heroku para permitir que cualquiera use su versión de Registro de aprendizaje. En la siguiente sección, completaremos algunas tareas más para finalizar el proceso de implementación y configurarlo para que continúe desarrollando el Registro de aprendizaje.

Refinando la implementación de Heroku

Ahora refinaremos la implementación creando un superusuario, tal como lo hicimos localmente. También haremos que el proyecto sea más seguro cambiando la configuración DEBUG a Falso, de modo que los usuarios no vean ninguna información adicional en los mensajes de error que podrían usar para atacar el servidor.

Crear un superusuario en Heroku

Ya has visto que podemos ejecutar comandos únicos usando la función heroku run dominio. Pero también puede ejecutar comandos abriendo una sesión de terminal Bash mientras está conectado al servidor Heroku usando el comando heroku run bash. Bash es el lenguaje que se ejecuta en muchos terminales Linux. Usaremos la sesión de terminal de Bash para crear un superusuario para que podamos acceder al sitio de administración en la aplicación en vivo:

```
(ll_env)learning_log$ heroku run bash
Ejecutando 'bash' en Ã secret-lowlands-82594... up, run.9858
u ~ $ ls
learning_log learning_logs manage.py Procfile requirements.txt runtime.txt
```

```
usuarios de archivos estáticos
v ~ $ python manage.py createsuperuser Nombre de usuario
(déjelo en blanco para usar 'u47318'): ll_admin Dirección de correo electrónico:
Contraseña:

Contraseña (nuevamente):
Superusuario creado con éxito. w ~ $ salir salir

(ll_env)registro_de_aprendizaje$
```

En u ejecutamos ls para ver qué archivos y directorios existen en el servidor, que deberían ser los mismos archivos que tenemos en nuestro sistema local. Puede navegar por este sistema de archivos como cualquier otro.

Los usuarios de Windows usarán los mismos comandos que se muestran aquí (como ls en lugar de dir), porque está ejecutando una terminal Linux a través de una conexión remota.

En v, ejecutamos el comando para crear un superusuario, que genera las mismas indicaciones que vimos en nuestro sistema local cuando creamos un superusuario en el Capítulo 18. Cuando haya terminado de crear el superusuario en esta sesión de terminal, ejecute el comando **exit** para regresar a la sesión de terminal de su sistema local w.

Ahora puede agregar */admin/* al final de la URL de la aplicación en vivo y inicie sesión en el sitio de administración. Para mí, la URL es *https:// secret-lowlands-82594 .herokuapp.com/ admin/.*

Si otros ya han comenzado a usar su proyecto, ¡tenga en cuenta que tendrá acceso a todos sus datos! No se lo tome a la ligera, y los usuarios seguirán confiando en usted con sus datos.

Creación de una URL fácil de usar en Heroku

Lo más probable es que desee que su URL sea más amigable y fácil de recordar que *https:// secret-lowlands-82594.herokuapp.com/.* Puede cambiar el nombre de la aplicación usando un solo comando:

```
(ll_env)learning_log$ heroku apps:rename learning-log
Cambiando el nombre de secret-lowlands-82594 a learning-log-2e... hecho
https://registro-de-aprendizaje.herokuapp.com/ | https://git.heroku.com/learning-log.git
Heroku remoto de Git actualizado
  Ã No olvide actualizar los controles remotos de git para todos los demás pagos locales de la aplicación.
(ll_env)registro_de_aprendizaje$
```

Puede usar letras, números y guiones al nombrar su aplicación y llamarla como quiera, siempre que nadie más haya reclamado el nombre. Esta implementación ahora se encuentra en *https:// learning-log.herokuapp.com/.* El proyecto ya no está disponible en la URL anterior; el comando apps:rename mueve completamente el proyecto a la nueva URL.

Cuando implementa su proyecto usando el servicio gratuito de Heroku, Heroku pone su implementación en modo de suspensión si no ha recibido ninguna solicitud después de una cierta cantidad de tiempo o si ha estado demasiado activo para el nivel gratuito. La primera vez que un usuario accede al sitio después de haber estado inactivo, tardará más en cargarse, pero el servidor responderá a las solicitudes posteriores con mayor rapidez. Así es como Heroku puede permitirse ofrecer implementaciones gratuitas.

Asegurar el proyecto en vivo

Existe un problema de seguridad evidente en la forma en que se implementa actualmente nuestro proyecto: la configuración DEBUG=True en *settings.py,* que proporciona mensajes de depuración cuando se producen errores. Las páginas de error de Django le brindan información de depuración vital cuando está desarrollando un proyecto; sin embargo, brindan demasiada información a los atacantes si los deja habilitados en un servidor en vivo.

Controlaremos si la información de depuración se muestra en el sitio activo configurando una variable de entorno. *Las variables de entorno* son valores establecidos en un entorno específico. Esta es una de las formas en que la información confidencial se almacena en un servidor, manteniéndola separada del resto del código del proyecto.

Modifiquemos *settings.py* para que busque una variable de entorno cuando el proyecto se ejecuta en Heroku:

configuración.py

```
--recorte--
# Configuración de Heroku.
importar django_heroku
django_heroku.settings(locales())

si os.environ.get('DEBUG') == 'VERDADERO':
    DEPURAR = Verdadero
elif os.environ.get('DEBUG') == 'FALSO':
    DEPURAR = Falso
```

El método os.environ.get() lee el valor asociado con una variable de entorno específica en cualquier entorno donde se ejecuta el proyecto. Si la variable que estamos solicitando está configurada, el método devuelve su valor; si no está configurado, el método devuelve Ninguno. El uso de variables de entorno para almacenar valores booleanos puede resultar confuso. En la mayoría de los casos, las variables de entorno se almacenan como cadenas y debe tener cuidado con esto. Considere este fragmento de una simple sesión de terminal de Python:

```
>>> bool('Falso')
Verdadero
```

El valor booleano de la cadena 'False' es True, porque cualquier cadena que no esté vacía se evalúa como True. Así que usaremos las cadenas 'VERDADERO' y 'FALSO', en mayúsculas, para aclarar que no estamos almacenando los valores booleanos verdaderos y falsos reales de Python. Cuando Django lee en la variable de entorno con la clave 'DEBUG' en Heroku, estableceremos DEBUG en True si el valor es 'TRUE' y False si el valor es 'FALSE'.

Confirmar y empujar cambios

Ahora debemos confirmar los cambios realizados en *settings.py* en el repositorio de Git y luego enviar los cambios a Heroku. Aquí hay una sesión de terminal que muestra este proceso:

```
u (ll_env)learning_log$ git commit -am "Establecer DEBUG basado en variables de entorno". [maestro
  3427244] Establecer DEPURACIÓN en función de las variables de entorno. 1 archivo cambiado, 4
  inserciones (+) v (ll_env)learning_log$ git status

En maestro de rama
nada que cometer, árbol de trabajo limpio
(ll_env)learning_log$
```

Emitimos el comando git commit con un mensaje de confirmación corto pero descriptivo u. Recuerde que el indicador -am se asegura de que Git confirme todos los archivos que han cambiado y registra el mensaje de registro. Git reconoce que un archivo ha cambiado y envía este cambio al repositorio.

En v, el estado muestra que estamos trabajando en la rama maestra del repositorio y que ahora no hay nuevos cambios para confirmar. Es esencial que verifique el estado de este mensaje antes de ingresar a Heroku. Si no ve este mensaje, algunos cambios no se han confirmado y esos cambios no se enviarán al servidor. Puede intentar emitir el comando de confirmación nuevamente, pero si no está seguro de cómo resolver el problema, lea el Apéndice D para comprender mejor cómo trabajar con Git.

Ahora empujemos el repositorio actualizado a Heroku:

```
(ll_env)learning_log$ git empujar heroku maestro
remoto: Fuente del edificio:
remoto:
remoto: -----> aplicación de Python detectada
remoto: -----> Requisitos de instalación con pip
--recorte--
remoto: -----> Iniciando...
remoto: Lanzado v6
remoto:            https://learning-log.herokuapp.com/ implementado en Heroku
remoto:
remoto: Verificando implementación... hecho.
A https://git.heroku.com/learning-log.git
   144f020..d5075a1 maestro -> maestro
(ll_env)registro_de_aprendizaje$
```

Heroku reconoce que el repositorio se ha actualizado y reconstruye el proyecto para asegurarse de que se hayan tenido en cuenta todos los cambios. No reconstruye la base de datos, por lo que no tendremos que ejecutar la migración para esta actualización.

Configuración de variables de entorno en Heroku

Ahora podemos establecer el valor que queremos para DEBUG en *settings.py* a través de Heroku. El comando heroku config:set establece una variable de entorno para nosotros:

```
(ll_env)learning_log$ heroku config:establecer DEBUG='FALSO'
Configurando DEBUG y reiniciando Ã learning-log... hecho, v7
DEPURACIÓN: FALSO
(ll_env)registro_de_aprendizaje$
```

Cada vez que establece una variable de entorno en Heroku, automáticamente reinicia el proyecto para que la variable de entorno pueda tener efecto.

Para verificar que la implementación es más segura ahora, ingrese la URL de su proyecto con una ruta que no hemos definido. Por ejemplo, intente visitar *http:// learning -log.herokuapp.com/ letmein/.* Debería ver una página de error genérica en su implementación en vivo que no brinda ninguna información específica sobre el proyecto. Si intenta la misma solicitud en la versión local de Learning Log en *http:// localhost:8000/ letmein/,* debería ver la página de error completa de Django. El resultado es perfecto: verá mensajes de error informativos cuando desarrolle más el proyecto en su propio sistema. Pero los usuarios del sitio en vivo no verán información crítica sobre el código del proyecto.

Si solo está implementando una aplicación y está solucionando problemas de la implementación inicial, puede ejecutar heroku config:set DEBUG='TRUE' y ver temporalmente un informe de error completo en el sitio en vivo. Solo asegúrese de restablecer el valor a 'FALSO' una vez que haya terminado de solucionar el problema. Además, tenga cuidado de no hacer esto una vez que los usuarios acceden regularmente a su sitio.

Creación de páginas de error personalizadas

En el Capítulo 19, configuramos el Registro de aprendizaje para que devuelva un error 404 si el usuario solicita un tema o una entrada que no le pertenece. Probablemente también haya visto unos 500 errores de servidor (errores internos) en este punto. Un error 404 generalmente significa que su código Django es correcto, pero el objeto solicitado no existe; un error 500 generalmente significa que hay un error en el código que ha escrito, como un error en una función en *views.py.* Actualmente, Django devuelve la misma página de error genérica en ambas situaciones. Pero podemos escribir nuestras propias plantillas de página de error 404 y 500 que coincidan con la apariencia general de Learning Log. Estas plantillas deben ir en el directorio raíz de plantillas.

Hacer plantillas personalizadas

En la carpeta *learning_log* más externa , crea una nueva carpeta llamada *templates.* Luego crea un nuevo archivo llamado *404.html;* la ruta a este archivo debe ser *learning_log / templates/ 404.html.* Aquí está el código para este archivo:

404.html

```
{% extiende "learning_logs/base.html" %}

{% bloque page_header%}
    <h2>El artículo que solicitaste no está disponible. (404)</h2>
{% endblock page_header%}
```

Esta sencilla plantilla proporciona la información genérica de la página de error 404, pero está diseñada para que coincida con el resto del sitio.

Crea otro archivo llamado *500.html* usando el siguiente código:

500.html
```
{% extiende "learning_logs/base.html" %}

{% bloque page_header%}
    <h2>Ha habido un error interno. (500)</h2>
{% endblock page_header%}
```

Estos nuevos archivos requieren un ligero cambio en *settings.py.*

configuración.py
```
--recorte--
PLANTILLAS = [
        {
                'BACKEND': 'django.template.backends.django.DjangoTemplates',
                'DIRS': [os.path.join(BASE_DIR, 'plantillas')],
                'APP_DIRS': Verdadero,
                --recorte--
        },
]
--recorte--
```

Este cambio le dice a Django que busque en el directorio raíz de plantillas las plantillas de la página de error.

Ver las páginas de error localmente

Si desea ver cómo se ven las páginas de error en su sistema antes de enviarlas a Heroku, primero deberá configurar Debug=False en su configuración local para suprimir las páginas de depuración predeterminadas de Django. Para hacerlo, realice el siguiente cambio en *settings.py* (asegúrese de estar trabajando en la parte de *settings.py* que se aplica al entorno local, no en la parte que se aplica a Heroku):

configuración.py
```
--recorte--
# ADVERTENCIA DE SEGURIDAD: ¡no ejecute con la depuración activada en producción!
DEPURAR = Falso
--recorte--
```

Ahora solicita un tema o entrada que no te pertenezca para ver la página de error 404. Para probar la página de error 500, solicite un tema o una entrada que no exista. Por ejemplo, la URL *http:// localhost:8000/ topics/ 999/* debería generar un error 500 a menos que ya haya generado 999 temas de ejemplo.

Cuando haya terminado de comprobar las páginas de error, establezca el valor local de DEBUG de nuevo en True para seguir desarrollando el registro de aprendizaje. (Asegúrese de no cambiar la forma en que se maneja DEBUG en la sección que administra la configuración en el entorno de Heroku).

La página de error 500 no mostrará ninguna información sobre el usuario que ha iniciado sesión, porque Django no envía ninguna información de contexto en la respuesta cuando hay un error del servidor.

Empujando los cambios a Heroku

Ahora necesitamos confirmar los cambios en la página de error que acabamos de hacer y enviarlos a Heroku:

```
u (ll_env)learning_log$ git add.
v (ll_env)learning_log$ git commit -am "Se agregaron páginas de error 404 y 500 personalizadas".
  3 archivos cambiados, 15 inserciones (+), 10 eliminaciones (-) modo
  de creación 100644 templates/404.html modo de creación 100644
  templates/500.html w (ll_env)learning_log$ git push heroku master

--recorte--
remoto: Verificando la implementación... hecho.
A https://git.heroku.com/learning-log.git d5075a1..4bd3b1c
   maestro -> maestro (ll_env)learning_log$
```

Emitimos los archivos git add en el proyecto, por lo que debemos decirle a Git que comience a rastrear estos en u porque creamos algunos nuevos. archivos. Luego, confirmamos los cambios v y enviamos el proyecto actualizado a Heroku w.

Ahora, cuando aparece una página de error, debe tener el mismo estilo que la resto del sitio, lo que hace que la experiencia del usuario sea más fluida cuando surgen errores.

Usando el método get_object_or_404()

En este punto, si un usuario solicita manualmente un tema o entrada que no existe, recibirá un error de servidor 500. Django intenta renderizar la página inexistente, pero no tiene suficiente información para hacerlo y el resultado es un error 500.
Esta situación se maneja con mayor precisión como un error 404, y podemos implementar este comportamiento usando la función de acceso directo de Django get_object_or_404().
Esta función intenta obtener el objeto solicitado de la base de datos, pero si ese objeto no existe, genera una excepción 404. Importaremos esta función a *views.py* y la usaremos en lugar de get():

vistas.py

```
desde django.shortcuts importar renderizar, redirigir, get_object_or_404
desde django.contrib.auth.decorators import login_required
--recorte--
@Necesario iniciar sesión
def tema (solicitud, topic_id):
    """Mostrar un solo tema y todas sus entradas."""
    tema = get_object_or_404(Tema, id=topic_id)
    # Asegúrese de que el tema pertenezca al usuario actual.
    --recorte--
```

Ahora, cuando solicita un tema que no existe (por ejemplo, *http://localhost:8000/ topics/ 999/),* verá una página de error 404. Para implementar este cambio, realice una nueva confirmación y luego envíe el proyecto a Heroku.

Desarrollo en curso

Es posible que desee desarrollar aún más el registro de aprendizaje después de su impulso inicial a un servidor en vivo o desarrollar sus propios proyectos para implementar. Hay un proceso bastante consistente para actualizar proyectos.

En primer lugar, realizará los cambios necesarios en su proyecto local. Si sus cambios dan como resultado nuevos archivos, agregue esos archivos al repositorio de Git usando el comando git add . (asegúrese de incluir el punto al final del comando). Cualquier cambio que requiera una migración de la base de datos necesitará este comando, porque cada migración genera un nuevo archivo de migración.

En segundo lugar, confirme los cambios en su repositorio usando git commit -am "commit message". Luego envíe sus cambios a Heroku usando el comando git push heroku master. Si migró su base de datos localmente, también deberá migrar la base de datos activa. Puede utilizar el comando único heroku run python manage.py migrate o abrir una sesión de terminal remota con heroku run bash y ejecutar el comando python manage.py migrate.

Luego visite su proyecto en vivo y asegúrese de que los cambios que espera ver hayan tenido efecto.

Es fácil cometer errores durante este proceso, así que no se sorprenda cuando algo salga mal. Si el código no funciona, revise lo que ha hecho e intente detectar el error. Si no puede encontrar el error o no sabe cómo deshacer el error, consulte las sugerencias para obtener ayuda en el Apéndice C. No se avergüence de pedir ayuda: todos los demás aprendieron a construir proyectos preguntando. las mismas preguntas que es probable que haga, por lo que alguien estará encantado de ayudarle. Resolver cada problema que surja lo ayuda a desarrollar sus habilidades de manera constante hasta que construya proyectos significativos y confiables y también responda las preguntas de otras personas.

La configuración SECRET_KEY

Django usa el valor de la configuración SECRET_KEY en *settings.py* para implementar una serie de protocolos de seguridad. En este proyecto, hemos enviado nuestro archivo de configuración al repositorio con la configuración SECRET_KEY incluida. Esto está bien para un proyecto de práctica, pero la configuración SECRET_KEY debe manejarse con más cuidado para un sitio de producción. Si crea un proyecto que está obteniendo un uso significativo, asegúrese de investigar cómo manejar su configuración SECRET_KEY de manera más segura.

Eliminación de un proyecto en Heroku

Es una buena práctica ejecutar el proceso de implementación varias veces con el mismo proyecto o con una serie de proyectos pequeños para familiarizarse con la implementación. Pero deberá saber cómo eliminar un proyecto que se ha implementado. Heroku también limita la cantidad de proyectos que puede alojar de forma gratuita, y no desea saturar su cuenta con proyectos de práctica.

Inicie sesión en el sitio web de Heroku *(https:// heroku.com/);* será redirigido a una página que muestra una lista de sus proyectos. Haga clic en el proyecto que desea eliminar. Verá una nueva página con información sobre el proyecto. Haga clic **en Configuración**

enlace y desplácese hacia abajo hasta que vea un enlace para eliminar el proyecto. Esta acción no se puede revertir, por lo que Heroku le pedirá que confirme la solicitud de eliminación ingresando manualmente el nombre del proyecto.

Si prefiere trabajar desde una terminal, también puede eliminar un proyecto emitiendo el comando destroy :

```
(ll_env)learning_log$ heroku apps:destroy --app appname
```

Aquí, *appname* es el nombre de su proyecto, que es algo así como secret-lowlands-82594 o learning-log si ha cambiado el nombre del proyecto. Se le pedirá que vuelva a ingresar el nombre del proyecto para confirmar la eliminación.

N ota La *eliminación de un proyecto en Heroku no afecta a la versión local del proyecto. Si nadie ha usado su proyecto implementado y solo está practicando el proceso de implementación, es perfectamente razonable eliminar su proyecto en Heroku y volver a implementarlo.*

Inténtalo tú mismo

20-3. Blog en vivo: implemente el proyecto de blog en el que ha estado trabajando en Heroku. Asegúrese de establecer DEBUG en Falso, para que los usuarios no vean las páginas de error completas de Django cuando algo sale mal.

20-4. Más 404: la función get_object_or_404() también debe usarse en las vistas new_entry() y edit_entry() . Realice este cambio, pruébelo ingresando una URL como http://localhost:8000/new_entry/999/, y verifique que vea un 404 error.

20-5. Registro de aprendizaje ampliado: agregue una función al registro de aprendizaje e impulse el cambio a su implementación en vivo. Pruebe con un cambio simple, como escribir más sobre el proyecto en la página de inicio. Luego intente agregar una característica más avanzada, como dar a los usuarios la opción de hacer público un tema. Esto requeriría un atributo llamado public como parte del modelo Topic (esto debe establecerse en False por defecto) y un elemento de formulario en la página new_topic que permite al usuario cambiar un tema de privado a público. Luego, deberá migrar el proyecto y revisar views.py para que cualquier tema que sea público también sea visible para los usuarios no autenticados. Recuerde migrar la base de datos en vivo después de haber enviado sus cambios a Heroku.

Resumen

En este capítulo, aprendió a dar a sus proyectos una apariencia simple pero profesional utilizando la biblioteca Bootstrap y la aplicación django-bootstrap4.
Al usar Bootstrap, los estilos que elija funcionarán de manera consistente en casi cualquier dispositivo que la gente use para acceder a su proyecto.

Aprendió sobre las plantillas de Bootstrap y usó la *plantilla estática de* la barra de navegación para crear una apariencia simple para el registro de aprendizaje. Usó un tron gigante para hacer que el mensaje de una página de inicio se destaque y aprendió a diseñar todas las páginas de un sitio de manera consistente.

En la parte final del proyecto, aprendió cómo implementar un proyecto en los servidores de Heroku para que cualquiera pueda acceder a él. Creó una cuenta de Heroku e instaló algunas herramientas que ayudan a administrar el proceso de implementación. Usó Git para enviar el proyecto de trabajo a un repositorio y luego empujó el repositorio a los servidores de Heroku. Finalmente, aprendió a comenzar a proteger su aplicación configurando DEBUG=False en el servidor en vivo.

Ahora que ha terminado el Registro de aprendizaje, puede comenzar a crear su propio proyectos Comience de manera simple y asegúrese de que el proyecto funcione antes de agregar complejidad. ¡Disfruta de tu aprendizaje continuo y buena suerte con tus proyectos!



Epílogo

¡Felicidades! Aprendió los conceptos básicos de Python y aplicó su conocimiento a proyectos significativos. Creó un juego, visualizó algunos datos e hizo una aplicación web. Desde aquí, puede ir en varias direcciones diferentes para continuar desarrollando sus habilidades de programación.

Primero, debe continuar trabajando en proyectos significativos que le interesen. La programación es más atractiva cuando está resolviendo problemas relevantes y significativos, y ahora tiene las habilidades para participar en una variedad de proyectos. Podrías inventar tu propio juego o escribir tu propia versión de un clásico juego de arcade. Es posible que desee explorar algunos datos que son importantes para usted y realizar visualizaciones que muestren patrones y conexiones interesantes. Puede crear su propia aplicación web o intentar emular una de sus aplicaciones favoritas.

Siempre que sea posible, invite a otras personas a probar sus programas. Si escribes un juego, deja que otras personas lo jueguen. Si hace una visualización, muéstresela a los demás y vea si tiene sentido para ellos. Si crea una aplicación web, impleméntela en línea e invite a otros a probarla. Escuche a sus usuarios e intente incorporar sus comentarios en sus proyectos; te convertirás en un mejor programador si lo haces.

Cuando trabaja en sus propios proyectos, se encontrará con problemas que son desafiantes, o incluso imposibles, de resolver por su cuenta. Siga encontrando formas de pedir ayuda y encuentre su propio lugar en la comunidad de Python. Únase a un grupo de usuarios de Python local o explore algunas comunidades de Python en línea. Considere asistir a una PyCon cerca de usted también.

Debe esforzarse por mantener un equilibrio entre trabajar en proyectos que le interesen y desarrollar sus habilidades de Python en general. Muchas fuentes de aprendizaje de Python están disponibles en línea y una gran cantidad de libros de Python están dirigidas a programadores intermedios. Muchos de estos recursos estarán accesibles para usted ahora que conoce los conceptos básicos y cómo aplicar sus habilidades. Trabajar con tutoriales y libros de Python se basará directamente en lo que aprendió aquí y profundizará su comprensión de la programación en general y de Python en particular. Luego, cuando vuelva a trabajar en proyectos después de concentrarse en aprender sobre Python, podrá resolver una variedad más amplia de problemas de manera más eficiente.

¡Felicitaciones por lo lejos que has llegado y buena suerte con tu aprendizaje continuo!

UN

Instalación y
Solución de problemas

Python tiene varias versiones y hay varias formas de configurarlo en cada sistema operativo. Use este apéndice para instalar Python si el enfoque del Capítulo 1 no funcionó o si desea instalar una versión de Python diferente a la que vino con su sistema.

Pitón en Windows

Las instrucciones del Capítulo 1 le muestran cómo instalar Python usando el instalador oficial en *https:// python.org/.* Si no pudo hacer que Python se ejecutara después de usar el instalador, las instrucciones de solución de problemas en esta sección deberían ayudarlo a poner en funcionamiento Python.

Encontrar el intérprete de Python

Si ingresó el comando simple python y obtiene un error, como *python no se reconoce como un comando interno o externo,* lo más probable es que

olvidó seleccionar la opción *Agregar Python a PATH* cuando ejecutó el instalador. En este caso, deberá decirle a Windows dónde encontrar el intérprete de Python. Para encontrarlo, abra su unidad C y busque la carpeta que comienza con el nombre *Python* (es posible que deba ingresar la palabra python en la barra de búsqueda del Explorador de Windows para encontrar la carpeta correcta, ya que podría estar anidada más abajo). Abra la carpeta y busque un archivo llamado *python* en minúsculas. Haga clic derecho en este archivo y elija **Propiedades;** la ruta a este archivo se enumerará bajo el encabezado *Ubicación.*

Para decirle a Windows dónde encontrar el intérprete, abra una ventana de terminal e ingrese la ruta seguida por el comando --version , así:

```
$ C:\Python37\python --versión
Pitón 3.7.2
```

Su ruta podría parecerse más a C:\Users\username\Programs \Python37\python en su sistema. Usando esta ruta, Windows debería ejecutar el intérprete de Python.

Agregar Python a su variable de ruta

Es molesto escribir la ruta completa cada vez que desea iniciar una sesión de terminal de Python, así que agreguemos la ruta al sistema para que pueda usar el comando python . Abra el **Panel de control de su sistema,** haga clic en **Sistema y seguridad** y luego haga clic en **Sistema.** Haga clic **en Configuración avanzada del sistema.** En la ventana que aparece, haga clic en **Variables de entorno.**

En el cuadro denominado *Variables del sistema,* busque una variable denominada Ruta. Hacer clic la palabra **Ruta** y luego haga clic en **Editar.** Debería ver una lista de ubicaciones en las que busca su sistema cuando busca programas. Haga clic en **Nuevo** y pegue la ruta a su archivo *python.exe* en el cuadro de texto que aparece. Si su sistema está configurado como el mío, sería:

```
C:\Python37
```

Tenga en cuenta que no incluimos el nombre del archivo *python.exe ;* eran simplemente diciéndole al sistema dónde buscarlo.

Cierra la ventana de tu terminal y abre una nueva. Al hacerlo, se cargará la nueva variable Path en su sesión de terminal. Ahora, cuando ingrese python --version, debería ver la versión de Python que acaba de agregar a su ruta variable. Ahora puede iniciar una sesión de terminal de Python simplemente ingresando python en un símbolo del sistema.

Si utiliza una versión anterior de Windows, es posible que vea un cuadro con la etiqueta Valor de variable *al hacer clic en* ***Editar.*** *Si ve este cuadro, use la tecla de flecha derecha para desplazarse completamente hacia la derecha. Tenga cuidado de no sobrescribir la variable existente; si lo hace, haga clic en* ***Cancelar*** *y vuelva a intentarlo. Agregue un punto y coma y la ruta a su* archivo python.exe *a la variable existente:*

```
%SystemRoot%\system32\...\System32\WindowsPowerShell\v1.0\;C:\Python37
```

Reinstalando Python

Si aún no puede ejecutar Python, a menudo desinstalar Python y volver a ejecutar el instalador resolverá cualquier problema que haya ocurrido en su primer intento.

Para hacer esto, abra el **Panel** de control de su sistema y haga clic en **Programas y características.** Desplácese hacia abajo hasta que vea la versión de Python que acaba de instalar y selecciónela. Haga clic en **Desinstalar/Cambiar** y luego haga clic en **Desinstalar** en el cuadro de diálogo que aparece. Luego, vuelva a ejecutar el instalador siguiendo las instrucciones del Capítulo 1, pero esta vez asegúrese de seleccionar la opción **Agregar Python a la RUTA** y cualquier otra configuración que sea relevante para su sistema. Si aún tiene problemas y no está seguro de dónde obtener ayuda, consulte las sugerencias en el Apéndice C.

Python en macOS

Las instrucciones de instalación en el Capítulo 1 usan el instalador oficial de Python en *https:// python.org/,* que le recomiendo que use a menos que tenga una razón específica para no hacerlo. Otro enfoque usa Homebrew, una herramienta que puede usar para instalar una variedad de software en macOS. Si ya está usando Homebrew y quiere usarlo para instalar Python, o si las personas con las que trabaja usan Homebrew y quiere una configuración similar a la que están usando, puede usar las siguientes instrucciones.

Instalar Homebrew

Homebrew depende de algunas de las herramientas de línea de comandos del paquete Xcode de Apple, por lo que primero deberá instalar las herramientas de línea de comandos de Xcode. Abra una terminal y ejecute este comando:

```
$ xcode-seleccionar --instalar
```

Haga clic en los cuadros de diálogo de confirmación que aparecen (esto puede llevar un tiempo, dependiendo de la velocidad de su conexión). A continuación, instale Homebrew ingresando el siguiente comando:

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Puede encontrar este comando en *https:// brew.sh/.* Asegúrese de incluir un espacio entre curl -fsSL y la URL.

La -e en este comando le dice a Ruby (el lenguaje de programación en el que está escrito Homebrew) que ejecute el código que se descarga aquí. Solo debe ejecutar comandos como este desde fuentes en las que confíe.

Para confirmar que Homebrew se instaló correctamente, ejecute este comando:

```
$ cerveza doctor
Su sistema está listo para preparar.
```

Este resultado significa que está listo para usar Homebrew para instalar paquetes en su sistema.

Instalación de Python

Para instalar la última versión de Python, ingrese el siguiente comando:

```
$ brew instalar python
```

Verifique qué versión se instaló usando este comando:

```
$ python3 --versión
Pitón 3.7.2
ps
```

Ahora puede iniciar una sesión de terminal de Python con el comando python3. También puede usar el comando python3 en su editor de texto para que ejecute programas con la versión de Python que acaba de instalar en lugar de la versión anterior del sistema. Si necesita ayuda para configurar Sublime Text para usar la versión que acaba de instalar, consulte las instrucciones en el Capítulo 1.

pitón es linux

Python se incluye de forma predeterminada en casi todos los sistemas Linux. Pero si la versión predeterminada es anterior a Python 3.6, debe instalar la última versión. Las siguientes instrucciones deberían funcionar para la mayoría de los sistemas basados en apt.

Usaremos un paquete llamado deadsnakes, que facilita la instalación de múltiples versiones de Python. Introduzca los siguientes comandos:

```
$ sudo add-apt-repositorio ppa:deadsnakes/ppa
$ sudo apt-get update $
sudo apt install python3.7
```

Estos comandos deberían instalar Python 3.7 en su sistema.

Ingrese el siguiente comando para iniciar una sesión de terminal que ejecute Python 3.7:

```
$ pitón3.7
>>>
```

También querrá usar este comando cuando configure su editor de texto y cuando ejecute programas desde la terminal.

Palabras clave de Python y funciones integradas

Python viene con su propio conjunto de palabras clave y funciones integradas. Es importante tener esto en cuenta cuando nombra variables: los nombres de las variables no pueden ser los mismos que estas palabras clave y no deben ser los mismos que los nombres de las funciones, o sobrescribirá las funciones.

En esta sección, enumeraremos las palabras clave de Python y los nombres de las funciones integradas, para que sepas qué nombres evitar.

Palabras clave de Python

Cada una de las siguientes palabras clave tiene un significado específico y verá un error si intenta usar alguna de ellas como nombre de variable.

Falso	esperar	demás	importar	aprobar
Ninguna	descanso excepto que la		en	elevar
cierto	clase finalmente es	continuar para		devolver
y	probar lambda			
como	def de no local mientras			
afirmar	del con	mundial	no	
asíncrono	rendimiento elif	si	o	

Funciones integradas de Python

No obtendrá un error si utiliza una de las siguientes funciones integradas fácilmente disponibles como nombre de variable, pero anulará el comportamiento de esa función:

abs()	delattr() hash() dict() help()	vista de memoria()	colocar()
all()	dir() hex() divmod() id()	min() siguiente()	setattr()
any()	enumerate() input() eval()	objeto() oct()	rodaja()
ascii()	exec() isinstance() filter()	abrir() ord() pow()	ordenado ()
bin()	issubclass() float( ) iter()	imprimir()	método estático()
bool()	formato() len() conjunto	propiedad()	cadena()
breakpoint()	congelado() lista()	rango() repr()	suma()
bytearray() bytes()		invertida() ronda()	súper()
callable() chr()			tupla()
classmethod()			tipo()
getattr() locals()			cuyo()
compile() complex()			Código Postal()
	globales() mapa()		__importar__()
	hasattr() max()		



B

Editores de texto e IDE

Los programadores dedican mucho tiempo a escribir, leer y editar código, y es esencial usar un editor de texto o un *entorno de desarrollo integrado (IDE)* para que este trabajo sea lo más eficiente posible. Un buen editor realizará tareas simples, como resaltar la estructura de su código para que pueda detectar errores comunes mientras trabaja. Pero no hará tanto como para distraerte de tu pensamiento. Los editores también tienen características útiles como sangrado automático, marcadores para mostrar la longitud de línea adecuada y atajos de teclado para operaciones comunes.

Un IDE es un editor de texto con una serie de otras herramientas incluidas, como depuradores interactivos e introspección de código. Un IDE examina su código a medida que lo ingresa e intenta aprender sobre el proyecto que está construyendo. Por ejemplo, cuando comienza a escribir el nombre de una función, un IDE puede mostrarle todos los argumentos que acepta la función. Este comportamiento puede ser muy útil cuando todo funciona y entiendes lo que estás viendo. Pero también puede ser abrumador como principiante y difícil de solucionar cuando no está seguro de por qué su código no funciona en el IDE.

Te animo a usar un editor de texto simple mientras aprendes a programar. Los editores de texto también suponen una carga mucho más ligera para su sistema; por lo tanto, si está trabajando en una máquina más antigua o con menos recursos, un editor de texto funcionará mejor que un IDE. Si ya está familiarizado con los IDE o si las personas que lo rodean usan un IDE y desea usar un entorno similar, pruébelos por todos los medios.

No se preocupe demasiado por elegir sus herramientas en este punto; su tiempo se dedicará mejor a profundizar en el idioma y trabajar en los proyectos que le interesan. Una vez que haya dominado los conceptos básicos, tendrá una mejor idea de qué herramientas funcionan para usted.

En este apéndice, configuraremos el editor de texto Sublime Text para ayudarlo a trabajar de manera más eficiente. También echaremos un breve vistazo a una serie de otros editores que podría considerar usar o ver que usan otros programadores de Python.

Personalización de la configuración de texto sublime

En el Capítulo 1, configuró Sublime Text para usar la versión de Python que desea al ejecutar sus programas. Ahora lo configuraremos para que realice algunas de las tareas mencionadas al principio de este apéndice.

Convertir tabulaciones en espacios

Si usa una combinación de tabuladores y espacios en su código, puede causar problemas en sus programas que son difíciles de diagnosticar. Para evitar esto, puede configurar Sublime Text para usar siempre espacios para la sangría, incluso cuando presiona la tecla de tabulación . Vaya a **VerÿSangría** y asegúrese de que la opción Sangría **usando espacios** esté seleccionada. Si no es así, selecciónelo. Además, asegúrese de que el **Ancho de la pestaña** se establece en **4** espacios.

Si ya tiene una combinación de tabulaciones y espacios en uno de sus programas, puede convertir todas las tabulaciones en espacios haciendo clic en **VerÿSangríaÿConvertir tabulaciones en espacios.** También puede acceder a esta configuración haciendo clic en **Espacios** en la parte inferior derecha de la ventana de Sublime Text.

Ahora puede usar la tecla de tabulación para sangrar líneas de código, y Sublime Text insertará espacios automáticamente para sangrar esas líneas.

Configuración del indicador de longitud de línea

La mayoría de los editores le permiten configurar una señal visual, generalmente una línea vertical, para mostrar dónde deben terminar sus líneas. En la comunidad de Python, la convención es limitar las líneas a 79 caracteres o menos. Para configurar esta función, seleccione **View4Ruler** y luego haga clic en **80.** Sublime Text colocará una línea vertical en la marca de 80 caracteres para ayudar a restringir las líneas de código a una longitud adecuada.

Sangría y eliminación de sangría de bloques de código

Para aplicar sangría a un bloque completo de código, resáltelo y seleccione **EditarÿLíneaÿIndentar** o presione ctrl-], o ÿ-] en macOS. Para eliminar la sangría de un bloque de código, haga clic en **Editarÿ Line4Unindent** o presione ctrl-[, o ÿ-[.

Comentar bloques de código

Para deshabilitar temporalmente un bloque de código, puede resaltar el bloque y comentarlo para que Python lo ignore. Seleccione **EditarÿComentario4Alternar comentario** (ctrl-/ o ÿ-/). Las líneas seleccionadas se comentarán con una almohadilla (#) sangrada al mismo nivel que la línea de código para indicar que no son comentarios normales. Cuando desee descomentar el bloque de código, resalte el bloque y vuelva a emitir el mismo comando.

Guardar su configuración

Algunas de las configuraciones mencionadas solo afectan el archivo actual en el que está trabajando. Para que su configuración afecte a todos los archivos que abre en Sublime Text, deberá definir su configuración de usuario. Seleccione **Sublime Text4Preferences4Settings** y busque el archivo *Preferences.sublime-settings – User.* Ingrese lo siguiente en este archivo:

```
{
    "gobernantes": [80],
    "traducir_tabs_a_espacios": verdadero
}
```

Guarde este archivo y la configuración de su regla y pestaña se aplicará a todos los archivos con los que trabaje en Sublime Text. Si agrega más configuraciones a este archivo, asegúrese de que cada línea termine con una coma excepto la última línea. Puede ver los archivos de configuración de otros usuarios en línea y personalizar su editor con la configuración que lo ayude a trabajar de manera más eficiente.

Otras personalizaciones

Puede personalizar Sublime Text de muchas maneras para ayudarlo a trabajar de manera aún más eficiente. Mientras explora los menús, esté atento a los atajos de teclado para los elementos de menú que usa con más frecuencia. Cada vez que usa un atajo de teclado en lugar de alcanzar el mouse o el panel táctil, se vuelve un poco más eficiente. No intentes aprender todo de golpe; solo trate de ser eficiente con las acciones que usa con más frecuencia y esté atento a otras funciones que puedan ayudarlo a desarrollar su propio flujo de trabajo.

Otros editores de texto e IDE

Escuchará y verá a personas que utilizan otros editores de texto. La mayoría de ellos se pueden configurar para ayudarlo de la misma manera que personalizó Sublime Text. Aquí hay una pequeña selección de editores de texto de los que quizás haya oído hablar.

INACTIVO

IDLE es un editor de texto que se incluye con Python. Es un poco menos intuitivo para trabajar que Sublime Text, pero verá referencias a él en otros tutoriales dirigidos a principiantes, por lo que es posible que desee probarlo.

geany

Geany es un editor de texto simple que le permite ejecutar todos sus programas directamente desde el editor. Muestra toda su salida en una ventana de terminal, lo que lo ayuda a sentirse cómodo usando terminales. Geany tiene una interfaz muy simple, pero es lo suficientemente poderosa como para que un número significativo de programadores experimentados aún la usen.

Emacs y Vim

Emacs y Vim son dos editores populares preferidos por muchos programadores experimentados porque están diseñados para usarse de modo que sus manos nunca tengan que dejar el teclado. Esto hace que escribir, leer y modificar código sea muy eficiente una vez que aprende cómo funciona el editor. También significa que ambos editores tienen una curva de aprendizaje bastante pronunciada. Vim está incluido en la mayoría de las máquinas Linux y macOS, y tanto Emacs como Vim se pueden ejecutar completamente dentro de una terminal. Por esta razón, a menudo se usan para escribir código en servidores a través de una sesión de terminal remota.

Los programadores a menudo recomendarán que los pruebes. Pero muchos los programadores competentes olvidan cuánto están tratando de aprender los nuevos programadores. Es beneficioso conocer estos editores, pero evite usarlos hasta que se sienta cómodo trabajando con código en un editor más simple que le permita concentrarse en aprender a programar en lugar de aprender a usar un editor.

Átomo

Atom es un editor de texto con algunas funciones que normalmente encontraría en un IDE. Puede abrir un archivo individual en el que esté trabajando, o puede abrir una carpeta de proyecto y Atom instantáneamente hará que todos los archivos en ese proyecto sean fácilmente accesibles. Atom está integrado con Git y GitHub, por lo que cuando comience a usar el control de versiones, podrá trabajar con repositorios locales y remotos desde su editor en lugar de tener que hacerlo en una terminal separada.

Atom también le permite instalar paquetes, por lo que puede extender su comportamiento ior de muchas maneras. Varios de estos paquetes incorporan un comportamiento que hace que Atom se comporte más como un IDE.

código de estudio visual

Visual Studio Code, también llamado VS Code, es otro editor que actúa más como un IDE. VS Code admite el uso eficiente de un depurador, tiene soporte de control de versiones integrado y también ofrece herramientas de finalización de código.

PyCharm

PyCharm es un IDE popular entre los programadores de Python porque fue creado para funcionar específicamente con Python. La versión completa requiere una suscripción paga, pero también está disponible una versión gratuita llamada PyCharm Community Edition que muchos desarrolladores encuentran útil.

PyCharm presenta un *linter,* que verifica que su estilo de codificación coincida con las convenciones comunes de Python y ofrece sugerencias cuando se desvía del formato normal de Python. También tiene un depurador integrado para ayudarlo a resolver errores de manera competente y modos que lo ayudan a trabajar de manera eficiente con varias bibliotecas populares de Python.

Cuadernos Jupyter

Jupyter Notebook es un tipo de herramienta diferente a los editores de texto o IDE tradicionales, ya que es una aplicación web construida principalmente con bloques; cada bloque es un bloque de código o un bloque de texto. Los bloques de texto se representan en Markdown, por lo que puede incluir un formato simple en sus bloques de texto.

Los portátiles Jupyter se desarrollaron para admitir el uso de Python en aplicaciones científicas, pero desde entonces se han ampliado para volverse útiles en una amplia variedad de situaciones. En lugar de simplemente escribir comentarios dentro de un archivo .py , puede escribir texto claro con formato simple, como encabezados, listas con viñetas e hipervínculos entre secciones de código. Cada bloque de código se puede ejecutar de forma independiente, lo que le permite probar pequeñas partes de su programa, o puede ejecutar todos los bloques de código a la vez. Cada bloque de código tiene su propia área de salida y puede activar o desactivar las áreas de salida según sea necesario.

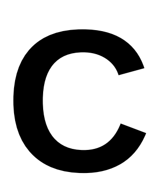
Jupyter Notebooks puede resultar confuso a veces debido a las interacciones entre diferentes celdas. Si define una función en una celda, esa función también está disponible para otras celdas. Esto es beneficioso la mayor parte del tiempo, pero puede resultar confuso en portátiles más largos y si no comprende completamente cómo funciona el entorno de Notebook.

Si está realizando algún trabajo científico o centrado en datos en Python, es casi seguro que verá Jupyter Notebooks en algún momento.



C

Obteniendo ayuda



Todo el mundo se atasca en algún momento cuando está aprendiendo a programar. Entonces, una de las habilidades más importantes que debe aprender un programador es cómo despegarse de manera eficiente. Este apéndice describe varias formas de ayudarlo a ponerse en marcha nuevamente cuando la programación se vuelve confusa.

Primeros pasos

Cuando esté atascado, su primer paso debe ser evaluar su situación. Antes de pedir ayuda a otra persona, responda claramente las siguientes tres preguntas:

- ¿Que estás tratando de hacer?
- ¿Qué has intentado hasta ahora?
- ¿Qué resultados has estado obteniendo?

Haga sus respuestas lo más específicas posible. Para la primera pregunta, explícito afirmaciones como "Estoy tratando de instalar la última versión de Python en mi computadora portátil con Windows 10" son lo suficientemente detalladas para que otros miembros de la comunidad de Python puedan ayudarlo. Declaraciones como "Estoy tratando de instalar Python" no brindan suficiente información para que otros puedan ofrecer mucha ayuda.

Su respuesta a la segunda pregunta debe proporcionar suficientes detalles para que no se le aconseje que repita lo que ya intentó: "Fui a *https:// python.org/ descargas/* e hizo clic en el botón Descargar para mi sistema. Luego ejecuté el instalador", es más útil que "Fui al sitio web de Python y descargué algo".

Para la tercera pregunta, es útil conocer los mensajes de error exactos que recibido para que pueda buscar una solución en línea o proporcionarlos cuando solicite ayuda.

A veces, solo responder estas tres preguntas antes de pedir ayuda a otros te permite ver algo que te estás perdiendo y despegarte sin tener que ir más lejos. Los programadores incluso tienen un nombre para esto: se llama *depuración de patitos de goma.* La idea es que si explica claramente su situación a un patito de goma (o cualquier objeto inanimado) y le hace una pregunta específica, a menudo podrá responder su propia pregunta. Algunas tiendas de programación incluso tienen un pato de goma real para animar a la gente a "hablar con el pato".

Vuelve a intentarlo

Simplemente volver al principio y volver a intentarlo puede ser suficiente para resolver muchos problemas. Digamos que está tratando de escribir un ciclo for basado en un ejemplo de este libro. Es posible que solo te hayas perdido algo simple, como dos puntos al final de la línea for . Repasar los pasos nuevamente podría ayudarlo a evitar repetir el mismo error.

Tomar un descanso

Si ha estado trabajando en el mismo problema durante un tiempo, tomar un descanso es una de las mejores tácticas que puede probar. Cuando trabajamos en la misma tarea durante largos períodos de tiempo, nuestros cerebros comienzan a concentrarse en una sola solución. Perdemos de vista las suposiciones que hemos hecho, y tomar un descanso nos ayuda a tener una nueva perspectiva del problema. No es necesario que sea un descanso prolongado, solo algo que lo saque de su mentalidad actual. Si ha estado sentado durante mucho tiempo, haga algo físico: camine un poco o salga un rato; tal vez beba un vaso de agua o coma un refrigerio ligero y saludable.

Si te sientes frustrado, podría valer la pena dejar tu trabajo por el dia. Una buena noche de sueño casi siempre hace que un problema sea más abordable.

Consulte los recursos de este libro

Los recursos en línea para este libro, disponibles en *https:// nostarch.com / pythoncrashcourse2e/,* incluir una serie de secciones útiles sobre la configuración

su sistema y trabajando en cada capítulo. Si aún no lo ha hecho, eche un vistazo a estos recursos y vea si hay algo que ayude a su situación.

Búsqueda en línea

Lo más probable es que alguien más haya tenido el mismo problema que usted tiene y haya escrito sobre él en línea. Las buenas habilidades de búsqueda y las consultas específicas lo ayudarán a encontrar los recursos existentes para resolver el problema al que se enfrenta. Por ejemplo, si tiene dificultades para instalar la última versión de Python en Windows 10, buscar *instalar python windows 10* y limitar los resultados a los recursos del último año podría llevarlo a una respuesta clara.

Buscar el mensaje de error exacto también puede ser extremadamente útil. Por ejemplo, supongamos que obtiene el siguiente error cuando intenta iniciar una sesión de terminal de Python:

```
> pitón
'python' no se reconoce como un comando interno o externo,
programa operable o archivo por lotes
>
```

Buscar la frase completa "python no se reconoce como un o comando externo" probablemente le dará un buen consejo.

Cuando comienza a buscar temas relacionados con la programación, algunos sitios aparecerá repetidamente. Describiré algunos de estos sitios brevemente, para que sepa cuán útiles pueden ser.

Desbordamiento de pila

Desbordamiento de pila *(https:// stackoverflow.com/)* es uno de los sitios de preguntas y respuestas más populares para programadores y, a menudo, aparecerá en la primera página de resultados de búsquedas relacionadas con Python. Los miembros publican preguntas cuando están atascados y otros miembros intentan dar respuestas útiles. Los usuarios pueden votar por las respuestas que les resulten más útiles, por lo que las mejores respuestas suelen ser las primeras que encontrará.

Muchas preguntas básicas de Python tienen respuestas muy claras en Stack Overflow, porque la comunidad las ha perfeccionado con el tiempo. Se alienta a los usuarios a publicar actualizaciones también, por lo que las respuestas tienden a mantenerse relativamente actualizadas. Al momento de escribir este artículo, se han respondido más de un millón de preguntas relacionadas con Python en Stack Overflow.

La documentación oficial de Python

La documentación oficial de Python *(https:// docs.python.org/)* es un poco más impredecible para los principiantes, porque su propósito es más documentar el lenguaje que brindar explicaciones. Los ejemplos en la documentación oficial deberían funcionar, pero es posible que no comprenda todo lo que se muestra. Aún así, es un buen recurso para verificar cuándo aparece en sus búsquedas y será más útil para usted a medida que continúe desarrollando su comprensión de Python.

Documentación de la Biblioteca Oficial

Si está utilizando una biblioteca específica, como Pygame, Matplotlib, Django, etc., los enlaces a la documentación oficial de ese proyecto a menudo aparecerán en las búsquedas, por ejemplo, *https:// docs.djangoproject.com/* es muy útil Si planea trabajar con alguna de estas bibliotecas, es una buena idea familiarizarse con su documentación oficial.

r/aprenderpython

Reddit se compone de una serie de subforos llamados *subreddits.* El subreddit *r/ learnpython (https:// reddit.com/ r/ learnpython/)* es bastante activo y solidario. Aquí puedes leer las preguntas de otros y publicar las tuyas.

Publicaciones de blog

Muchos programadores mantienen blogs y comparten publicaciones sobre las partes del lenguaje con las que están trabajando. Debe hojear los primeros comentarios en una publicación de blog para ver qué reacciones han tenido otras personas antes de tomar cualquier consejo. Si no aparecen comentarios, toma la publicación con pinzas. Es posible que nadie más haya verificado el consejo.

Internet Relay Chat

Muchos programadores interactúan en tiempo real a través de Internet Relay Chat (IRC). Si está atascado en un problema y la búsqueda en línea no proporciona respuestas, preguntar en un canal de IRC podría ser una buena opción. La mayoría de las personas que pasan el rato en estos canales son amables y serviciales, especialmente si puede ser específico sobre lo que está tratando de hacer, lo que ya ha intentado y los resultados que está obteniendo.

Crear una cuenta de IRC

Para crear una cuenta en IRC, vaya a *https:// webchat.freenode.net/.* Elija un apodo, complete el cuadro CAPTCHA y haga clic en **Conectar.** Verá un mensaje de bienvenida al servidor IRC de freenode. En el cuadro en la parte inferior de la ventana, ingrese el siguiente comando:

/msg nickserv registrar *contraseña correo electrónico*

Ingrese su propia contraseña y dirección de correo electrónico en lugar de *la contraseña* y el *correo electrónico.* Elija una contraseña que no use para ninguna otra cuenta. Recibirás un correo electrónico con instrucciones para verificar tu cuenta. El correo electrónico le proporcionará un comando como este:

/msg nickserv verificar registrar *apodo* código_verificación

Pegue esta línea en el sitio de IRC con el *apodo* como el nombre que eligió. lier y un valor para *código_verificación.* Ahora estás listo para unirte a un canal.

Si tiene problemas para iniciar sesión en su cuenta en algún momento, puede ejecutar el siguiente comando:

/msg nickserv identificar *apodo contraseña*

Reemplace el *apodo* y *la contraseña* con su propio apodo y contraseña. Esto lo autenticará en la red y podrá acceder a los canales que requieren un apodo autenticado.

Canales para unirse

Para unirse al canal principal de Python, ingrese /**join #python** en el cuadro de entrada. Verá una confirmación de que se unió al canal y alguna información general sobre el canal.

El canal *##learnpython* (con dos hashtags) también suele ser bastante activo. Este canal está asociado con *https:// reddit.com/ r/ learnpython/,* por lo que también verá mensajes sobre publicaciones en *r/ learnpython .* Es posible que desee unirse al canal *#django* si está trabajando en aplicaciones web.

Después de unirse a un canal, puede leer las conversaciones que tienen otras personas y también hacer sus propias preguntas.

Cultura del IRC

Para obtener ayuda efectiva, debe conocer algunos detalles sobre la cultura de IRC. Centrarse en las tres preguntas al comienzo de este apéndice definitivamente lo guiará hacia una solución exitosa. Las personas estarán felices de ayudarlo si puede explicar con precisión lo que está tratando de hacer, lo que ya ha intentado y los resultados exactos que está obteniendo. Si necesita compartir código o salida, los miembros de IRC usan sitios externos creados para este propósito, como *https:// bpaste.net/ +python.* (Aquí es donde #python lo envía a compartir código y salida). Esto evita que los canales se inunden con código y también hace que sea mucho más fácil leer el código que la gente comparte.

Ser paciente siempre hará que la gente esté más dispuesta a ayudarte. Pregunta tu pregunta de manera concisa y luego espera a que alguien responda. A menudo, las personas están en medio de muchas conversaciones, pero por lo general alguien se dirigirá a usted en un tiempo razonable. Si hay pocas personas en el canal, puede llevar un tiempo obtener una respuesta.

Flojo

Slack es como una reinvención moderna de IRC. A menudo se usa para comunicaciones internas de la empresa, pero también hay muchos grupos públicos a los que puede unirse. Si desea consultar los grupos de Python Slack, comience con *https:// pyslackers.com/.* Haga clic en el enlace de **Slack** en la parte superior de la página e ingrese su dirección de correo electrónico para recibir una invitación.

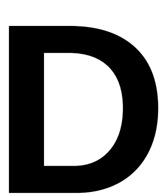
Una vez que esté en el espacio de trabajo de Python Developers, verá una lista de canales. Haga clic en **Canales** y luego elija los temas que le interesen. Es posible que desee comenzar con los canales *#learning_python* y *#django .*

Discordia

Discord es otro entorno de chat en línea con una comunidad de Python donde puede solicitar ayuda y seguir debates relacionados con Python.

Para comprobarlo, diríjase a *https:// pythondiscord.com/* y haga clic en **Chatear ahora** Enlace. Debería ver una pantalla con una invitación generada automáticamente; haz clic en **Aceptar invitación.** Si ya tiene una cuenta de Discord, puede iniciar sesión con su cuenta existente. Si no tiene una cuenta, ingrese un nombre de usuario y siga las indicaciones para completar su registro en Discord.

Si es la primera vez que visita Python Discord, deberá aceptar las reglas de la comunidad antes de participar por completo. Una vez que hayas hecho eso, puedes unirte a cualquiera de los canales que te interesen. Si está buscando ayuda, asegúrese de publicar en uno de los canales de ayuda de Python.



Uso de G it para el control de versiones

El software de control de versiones le permite tomar instantáneas de un proyecto siempre que esté en funcionamiento. Cuando haces cambios a un proyecto, por ejemplo, cuando implementa una nueva función, puede volver a un estado de trabajo anterior si el estado actual del proyecto no funciona bien.

El uso del software de control de versiones le brinda la libertad de trabajar en mejoras y cometer errores sin preocuparse por arruinar su proyecto. Esto es especialmente crítico en proyectos grandes, pero también puede ser útil en proyectos más pequeños, incluso cuando esté trabajando en programas contenidos en un solo archivo.

En este apéndice, aprenderá a instalar Git y a usarlo para el control de versiones en los programas en los que está trabajando ahora. Git es el software de control de versiones más popular en uso hoy en día. Muchas de sus herramientas avanzadas ayudan a los equipos a colaborar en proyectos grandes, pero sus funciones más básicas también funcionan bien para desarrolladores independientes. Git implementa el control de versiones mediante el seguimiento de los cambios realizados en cada archivo de un proyecto; si comete un error, simplemente puede volver a un estado previamente guardado.

Instalando Git

Git se ejecuta en todos los sistemas operativos, pero existen diferentes enfoques para instalarlo en cada sistema. Las siguientes secciones proporcionan instrucciones específicas para cada sistema operativo.

Instalación de Git en Windows

Puede descargar un instalador para Git en *https:// git-scm.com/.* Debería ver un enlace de descarga para un instalador apropiado para su sistema.

Instalación de Git en macOS

Es posible que Git ya esté instalado en su sistema, así que intente ejecutar el comando **git --version.** Si ve un resultado que enumera un número de versión específico, Git está instalado en su sistema. Si ve un mensaje que le pide que instale o actualice Git, simplemente siga las instrucciones en pantalla.

También puede visitar *https:// git-scm.com/,* donde deberías ver una descarga enlace para un instalador apropiado para su sistema.

Instalación de Git en Linux

Para instalar Git en Linux, ingrese el siguiente comando:

```
$ sudo apt install git-all
```

Eso es todo. Ahora puedes usar Git en tus proyectos.

Configuración de Git

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando solo una persona está trabajando en el proyecto. Para hacer esto, Git necesita saber su nombre de usuario y correo electrónico. Debe proporcionar un nombre de usuario, pero puede inventar una dirección de correo electrónico falsa:

```
$ git config --usuario global.nombre "nombre de usuario"
$ git config --usuario global.email "nombredeusuario@ejemplo.com"
```

Si olvida este paso, Git le pedirá esta información cuando realice su primera confirmación.

hacer un proyecto

Hagamos un proyecto para trabajar. Cree una carpeta en algún lugar de su sistema llamada *git_practice.* Dentro de la carpeta, haz un programa Python simple:

hola_git.py

```
imprimir("¡Hola, mundo de Git!")
```

Usaremos este programa para explorar la funcionalidad básica de Git.

Ignorar archivos

Los archivos con la extensión .pyc se generan automáticamente a partir de archivos .py , por lo que no necesitamos que Git realice un seguimiento de ellos. Estos archivos se almacenan en un directorio llamado *__pycache__*. Para decirle a Git que ignore este directorio, crea un archivo especial llamado .gitignore, *con* un punto al comienzo del nombre del archivo y sin extensión de archivo, y agrega la siguiente línea:

.gitignore

```
__pycache__/
```

Este archivo le dice a Git que ignore cualquier archivo en el directorio *__pycache__* . Usando un El archivo .gitignore mantendrá su proyecto ordenado y será más fácil trabajar con él.

Es posible que deba modificar la configuración de su editor de texto para que muestre los archivos ocultos para poder abrir .gitignore. Algunos editores están configurados para ignorar los nombres de archivo que comienzan con un punto.

Inicializar un repositorio

Ahora que tiene un directorio que contiene un archivo de Python y un archivo .gitignore , puede inicializar un repositorio de Git. Abra una terminal, navegue hasta el *git* carpeta *_practice* y ejecute el siguiente comando:

```
git_practice$ git init
Repositorio Git vacío inicializado en git_practice/.git/
git_practice$
```

El resultado muestra que Git ha inicializado un repositorio vacío en *git _práctica.* Un *repositorio* es el conjunto de archivos en un programa que Git rastrea activamente. Todos los archivos que usa Git para administrar el repositorio se encuentran en el directorio oculto .git, con el que no necesitará trabajar en absoluto. Simplemente no elimine ese directorio, o perderá el historial de su proyecto.

Comprobación del estado

Antes de hacer nada más, veamos el estado del proyecto:

```
git_practice$ git status u En
rama maestra

No hay compromisos todavía

v Archivos sin
    seguimiento: (use "git add <archivo>..." para incluir en lo que se confirmará)

    .gitignore
    hola_git.py

w no se agregó nada para confirmar, pero hay archivos sin rastrear presentes (use "git add" para rastrear)
git_practice$
```

En Git, una *rama* es una versión del proyecto en el que estás trabajando; aquí puedes ver que estamos en una sucursal llamada master u. Cada vez que verifique el estado de su proyecto, debería mostrar que está en el maestro de sucursales. Luego verá que estamos a punto de hacer la confirmación inicial. Una *confirmación* es una instantánea del proyecto en un momento determinado.

Git nos informa que los archivos sin rastrear están en el proyecto v, porque aún no le hemos dicho qué archivos rastrear. Luego, se nos dice que no se agregó nada a la confirmación actual, pero que hay archivos sin seguimiento que quizás deseemos agregar al repositorio w.

Adición de archivos al repositorio

Agreguemos los dos archivos al repositorio y verifiquemos el estado nuevamente:

```
u git_practice$ git add. v
git_practice$ git status En
  maestro de rama

  No hay compromisos todavía

  Cambios a confirmar: (use
    "git rm --cached <archivo>..." para quitar la preparación)

w nuevo archivo: .gitignore
    nuevo archivo: hello_git.py

  git_practice$
```

El comando git agregar. agrega todos los archivos dentro de un proyecto que aún no están siendo rastreados al repositorio u. No compromete los archivos; simplemente le dice a Git que comience a prestarles atención. Cuando verificamos el estado del proyecto ahora, podemos ver que Git reconoce algunos cambios que deben confirmarse v. La etiqueta *nuevo archivo* significa que estos archivos se agregaron recientemente al repositorio w.

Hacer un compromiso

Hagamos el primer commit:

```
u git_practice$ git commit -m "Proyecto iniciado". v
[master (root-commit) ee76419] Proyecto iniciado. w 2
archivos cambiados, 4 inserciones (+) modo de creación
    100644 .gitignore modo de creación 100644 hello_git.py
    x git_practice$ git status En branch master nada que
confirmar, árbol de trabajo limpio git_practice$
```

Emitimos el comando git commit -m *"message"* u para tomar una instantánea del proyecto. El indicador -m le dice a Git que registre el mensaje que sigue ("Proyecto iniciado") en el registro del proyecto. La salida muestra que estamos en el maestro branch v y que dos archivos han cambiado w.

Cuando verificamos el estado ahora, podemos ver que estamos en el maestro rama, y tenemos un árbol de trabajo limpio x. Este es el mensaje que desea ver cada vez que confirma un estado de trabajo de su proyecto. Si recibe un mensaje diferente, léalo detenidamente; es probable que haya olvidado agregar un archivo antes de realizar una confirmación.

Comprobación del registro

Git mantiene un registro de todas las confirmaciones realizadas en el proyecto. Revisemos el registro:

```
git_practice$ git log
commit a9d74d87f1aa3b8f5b2688cb586eac1a908cfc7f (HEAD -> maestro)
Autor: Eric Matthes <eric@example.com> Fecha:
lun 21 de enero 21:24:28 2019 -0900

    Proyecto iniciado.
git_practice$
```

Cada vez que realiza una confirmación, Git genera una ID de referencia única de 40 caracteres. Registra quién realizó la confirmación, cuándo se realizó y el mensaje grabado. No siempre necesitará toda esta información, por lo que Git ofrece una opción para imprimir una versión más simple de las entradas de registro:

```
git_practice$ git log --pretty=oneline
ee76419954379819f3f2cacafd15103ea900ecb2 (HEAD -> master) Proyecto iniciado.
git_practice$
```

El indicador --pretty=oneline proporciona las dos piezas de información más importantes: el ID de referencia de la confirmación y el mensaje registrado para la confirmación.

El segundo compromiso

Para ver el poder real del control de versiones, necesitamos hacer un cambio en el proyecto y confirmar ese cambio. Aquí solo agregaremos otra línea a *hola_git.py:*

hola_git.py

```
imprimir("¡Hola, mundo de Git!")
imprimir("Hola a todos")
```

Cuando comprobemos el estado del proyecto, veremos que Git ha notado el archivo que cambió:

```
git_practice$ git status u En
rama maestra
```

```
Cambios no preparados para
    confirmación: (use "git add <archivo>..." para actualizar lo que se
    confirmará) (use "git checkout -- <archivo>..." para descartar cambios en el directorio de trabajo)

v modificado: hello_git.py

w no se agregaron cambios para confirmar (use "git add" y/o "git commit -a")
git_practice$
```

Vemos la rama en la que estamos trabajando u, el nombre del archivo que se modificó v, y que no se han confirmado cambios w. Confirmemos el cambio y verifiquemos el estado nuevamente:

```
u git_practice$ git commit -am "Saludo extendido". [master
 51f0fe5] Saludo extendido. 1 archivo modificado, 1 inserción
  (+), 1 eliminación (-)
v git_practice$ git status En
branch master nada que
confirmar, árbol de trabajo limpio w git_practice$
git log --pretty=oneline
51f0fe5884e045b91c12c5449fabf4ad0eef8e5d (HEAD -> master) Saludo extendido.
ee76419954379819f3f2cacafd15103ea900ecb2 Proyecto iniciado. git_practice$
```

Hacemos una nueva confirmación, pasando las banderas -am cuando usamos el comando git commit u. El indicador -a le dice a Git que agregue todos los archivos modificados en el repositorio a la confirmación actual. (Si crea nuevos archivos entre confirmaciones, simplemente vuelva a emitir el comando git add para incluir los nuevos archivos en el repositorio). El indicador -m le dice a Git que registre un mensaje en el registro para este comprometerse.

Cuando verificamos el estado del proyecto, vemos que nuevamente tenemos un directorio de trabajo limpio v. Finalmente, vemos las dos confirmaciones en el registro w.

Revertir un cambio

Ahora veamos cómo abandonar un cambio y volver al estado de trabajo anterior. Primero, agrega una nueva línea a *hello_git.py:*

hola_git.py

```
imprimir("¡Hola, mundo de Git!")
imprimir("Hola a todos")

print("¡Oh, no, rompí el proyecto!")
```

Guarde y ejecute este archivo.

Comprobamos el estado y vemos que Git nota este cambio:

```
git_practice$ estado de git
En maestro de rama
Cambios no organizados para la
    confirmación: (use "git add <file>..." para actualizar lo que se confirmará)
```

```
(use "git checkout -- <archivo>..." para descartar cambios en el directorio de trabajo)

has modificado: hello_git.py

no se agregaron cambios para confirmar (use "git add" y/o "git commit -a")
git_practice$
```

Git ve que modificamos *hello_git.py* u, y podemos confirmar el cambio si queremos. Pero esta vez, en lugar de confirmar el cambio, volveremos a la última confirmación cuando sabíamos que nuestro proyecto estaba funcionando. No le haremos nada a *hello_git.py:* no eliminaremos la línea ni usaremos la función Deshacer en el editor de texto. En su lugar, ingrese los siguientes comandos en su terminal sesión:

```
git_practice$ git checkout.
git_practice$ git status En branch
master nada que confirmar, árbol
de trabajo limpio git_practice$
```

El comando git checkout te permite trabajar con cualquier confirmación anterior. El comando git checkout. abandona cualquier cambio realizado desde la última confirmación y restaura el proyecto al último estado confirmado.

Cuando regrese a su editor de texto, verá que *hello_git.py* ha vuelto a cambiar a esto:

```
imprimir("¡Hola, mundo de Git!")
imprimir("Hola a todos")
```

Aunque volver a un estado anterior puede parecer trivial en este proyecto simple, si estuviéramos trabajando en un proyecto grande con docenas de archivos modificados, todos los archivos que habían cambiado desde la última confirmación se revertirían. Esta función es increíblemente útil: puede realizar tantos cambios como desee al implementar una nueva función y, si no funcionan, puede descartarlos sin afectar el proyecto. No tiene que recordar esos cambios y deshacerlos manualmente. Git hace todo eso por ti.

Puede que tenga que actualizar el archivo en su editor para ver la versión anterior.

Comprobación de confirmaciones anteriores

Puede verificar cualquier compromiso en su registro, no solo el más reciente, al incluir los primeros seis caracteres de la ID de referencia en lugar de un punto. Al verificar y revisar una confirmación anterior, puede volver a la confirmación más reciente o abandonar su trabajo reciente y retomar el desarrollo de la confirmación anterior:

```
git_practice$ git log --pretty=oneline
51f0fe5884e045b91c12c5449fabf4ad0eef8e5d (HEAD -> master) Saludo extendido.
```

```
ee76419954379819f3f2cacafd15103ea900ecb2 Proyecto iniciado.
git_practice$ git checkout ee7641 Nota: revisando 'ee7641'.


u Está en estado de 'CABEZA desconectada'. Puede mirar a su alrededor, realizar cambios
   experimentales y confirmarlos, y puede descartar cualquier confirmación que realice en este estado
   sin afectar ninguna rama al realizar otra comprobación.

   Si desea crear una nueva rama para retener las confirmaciones que creó, puede hacerlo (ahora
   o más tarde) usando -b con el comando de pago nuevamente. Ejemplo:

     git checkout -b <nuevo-nombre-de-sucursal>

   HEAD está ahora en ee7641... Proyecto iniciado.
   git_practice$
```

Cuando verifica una confirmación anterior, abandona la rama maestra e ingresa a lo que Git se refiere como un estado *HEAD independiente* u. *HEAD* es el estado comprometido actual del proyecto; estás *desconectado* porque has dejado una rama con nombre (maestro, en este caso).

Para volver a la rama maestra , échale un vistazo:

```
git_practice$ git checkout master La
posición anterior de HEAD era ee76419 Proyecto iniciado.
Cambiado a la rama 'maestro'
git_practice$
```

Este comando lo lleva de vuelta a la rama maestra . A menos que desee trabajar con algunas funciones más avanzadas de Git, es mejor no realizar ningún cambio en su proyecto cuando haya verificado una confirmación anterior. Sin embargo, si eres el único que trabaja en un proyecto y deseas descartar todas las confirmaciones más recientes y volver a un estado anterior, puedes restablecer el proyecto a una confirmación anterior. Trabajando desde la rama maestra , ingrese lo siguiente:

```
u git_practice$ git estado
   En el maestro de la

   rama, no hay nada que confirmar, el directorio de trabajo está limpio
v git_practice$ git log --pretty=oneline
   51f0fe5884e045b91c12c5449fabf4ad0eef8e5d (HEAD -> master) Saludo extendido.
   ee76419954379819f3f2cacafd15103ea900ecb2 Proyecto iniciado. w
git_practice$ git reset --hard ee76419 HEAD ahora está en ee76419 Proyecto
   iniciado. x git_practice$ git status En branch master nada que confirmar,
directorio de trabajo limpio


y git_practice$ git log --pretty=oneline
   ee76419954379819f3f2cacafd15103ea900ecb2 (HEAD -> master) Proyecto iniciado. git_practice$
```

Primero verificamos el estado para asegurarnos de que estamos en la rama maestra u. Cuando miramos el registro, vemos ambas confirmaciones v. Luego emitimos el git

reset --hard comando con los primeros seis caracteres de la ID de referencia de el compromiso al que queremos volver permanentemente w. Comprobamos el estado de nuevo y vemos que estamos en la rama maestra sin nada que cometer x. Cuando volvemos a mirar el registro, vemos que estamos en la confirmación que queríamos comenzar de nuevo desde y.

Borrando el Repositorio

A veces estropearás el historial de tu repositorio y no sabrás cómo recuperarlo. Si esto sucede, primero considere pedir ayuda usando los métodos discutidos en el Apéndice C. Si no puede arreglarlo y está trabajando en un proyecto en solitario, puede continuar trabajando con los archivos pero deshacerse del historial del proyecto eliminando el directorio .git . Esto no afectará el estado actual de ninguno de los archivos, pero eliminará todas las confirmaciones, por lo que no podrá verificar ningún otro estado del proyecto.

Para hacer esto, abra un explorador de archivos y elimine el repositorio .git o elimínelo desde la línea de comandos. Luego, deberá comenzar de nuevo con un repositorio nuevo para comenzar a rastrear sus cambios nuevamente. Así es como se ve todo este proceso en una sesión de terminal:

```
u git_practice$ git status En branch
  master nada que confirmar,
  directorio de trabajo limpio v git_practice$ rm -rf .git

w git_practice$ git status fatal: No
  es un repositorio de git (ni ninguno de los directorios principales): .git x git_practice$ git init

  Repositorio Git vacío inicializado en git_practice/.git/
y git_practice$ git estado
  En maestro de rama

  No hay compromisos todavía

  Archivos sin seguimiento:
    (use "git add <file>..." para incluir en lo que se confirmará)

    .gitignore
    hola_git.py

  no se agregó nada para confirmar, pero hay archivos sin rastrear presentes (use "git add" para rastrear)
z git_practice$ git agregar.
  git_practice$ git commit -m "Volver a empezar". [master
  (root-commit) 6baf231] Empezar de nuevo. 2 archivos
   cambiados, 4 inserciones (+) modo de creación
   100644 .gitignore modo de creación 100644 hello_git.py
   { git_practice$ git status En branch master nada que
confirmar, árbol de trabajo clean git_practice$
```

Primero verificamos el estado y vemos que tenemos un árbol de trabajo limpio u. Luego usamos el comando rm -rf .git para eliminar el directorio .git (rmdir /s .git en Windows) v. Cuando verificamos el estado después de eliminar la carpeta .git , se nos dice que este no es un repositorio de Git w. Toda la información que usa Git para rastrear un repositorio se almacena en la carpeta .git , por lo que eliminarla elimina todo el repositorio.

Entonces podemos usar git init para iniciar un nuevo repositorio x. Verificar el estado muestra que estamos de vuelta en la etapa inicial, esperando el primer compromiso. Agregamos los archivos y hacemos el primer commit z. Verificar el estado ahora nos muestra que estamos en la nueva rama maestra sin nada que confirmar {.

Usar el control de versiones requiere un poco de práctica, pero una vez que comience a usarlo, nunca querrá volver a trabajar sin él.

Índice

Símbolos

+ (suma), 26 *
(asterisco) operador, 147 {}
(llaves), 92, 105 / (división), 26 **
(doble asterisco) operador, 149
== (operador de igualdad), 72–73 **
(exponente), 26 // (división de piso), 260 >
(mayor que), 75 >= (mayor o igual que), 75 #
(marca de almohadilla), para comentarios, 29 !
= (operador de desigualdad), 74 < (menor
que), 75 <= (menor o igual que), 75 %
(operador módulo), 116–117, 122 *
(multiplicación), 26 \n (nueva línea), 22 ! (no),
74 += operador, 115 [] (corchetes), 34 -
(resta), 26 \t (tabulador), 22

UN

adición (+), 26 alias,
152 *alice.py,* 197–
199 proyecto Alien
Invasion. *Véase también* extraterrestres
de Pygame

comprobando bordes,
266 colisiones, con balas,
268–269, 291–292
colisiones, con barco, 272–275
controlando la dirección de la flota,
266 creando un extraterrestre,
256

creando la flota, 258–264 dejando
caer la flota, 267 llegando al final
de la pantalla, 276 reconstruyendo
la flota, 270 balas, 246–252
colisiones, con extraterrestres,
268–269, 291–292 borrando viejos,
250 disparando, 249 limitando el
número de , 251 haciendo
más grande, 270 configuraciones,
247 acelerando, 271 clases

Extranjero, 256–258
Viñeta, 247–248
Botón, 280–281
Estadísticas del juego, 273
Marcador, 288–289
Ajustes, 231
Barco, 233–235

finalizando el juego, 276
archivos

alien_invasion.py, 229
bullet.py, 247 *button.py,*
280 *game_stats.py,* 273
scoreboard.py, 288
settings.py, 231 *ship.bmp,*
233 inicializando
configuraciones dinámicas,
286

agregar

niveles, 285–287
modificar los ajustes
de velocidad,
285 restablecer la velocidad,
287 planificar, 228

Proyecto Alien Invasion, *continuar* añadiendo
el botón Reproducir, 280–285
desactivar, 284 dibujar, 281 ocultar
el cursor del ratón, 284 reiniciar el
juego, 283 iniciar el juego, 283
puntuar, 288–300 todos los
aciertos, 292 puntuación alta,
294 aumentar los valores de los
puntos , 292 nivel, 296–298
número de barcos, 298–300 reinicio,
291 redondeo y formateo, 293–
294 atributo de puntuación , 288
actualización, 291 configuración,
almacenamiento, 231 ajuste de
velocidad del barco, 241–243
movimiento continuo, 239–241
búsqueda una imagen, 232 rango
límite, 243 *diversiones_park.py,*
80–83 y palabra clave, 75 API
(programación de aplicaciones
interfaz), 359
llamadas, 359 para GitHub,
371 para Hacker News, 372–
375 procesamiento de respuestas,
361–365 límites de velocidad, 365
solicitud de datos, 360 visualización de
resultados, 366–371 apostrophe.py , 24
método append() , 37 aplicación interfaz
de programación
Consulte Argumentos de
la API (interfaz de programación
de aplicaciones), 131–137. *Ver también*
funciones: aritmética de
argumentos, 26 como palabra clave, 152
métodos de afirmación, 212, 216
asterisco (*) operador, 147
atributos, 159. *Ver también* clases:
atributos

B

banned_users.py, 77
bicycles.py, 34–36
Archivos de imagen .bmp (mapa de
bits), 232 cuerpo de una función, 130 de
un archivo HTML, 440
Valores booleanos, 77, 456
Bootstrap, 438–447 llaves
({}), 92, 105 funciones
integradas, 471

C

CamelCase, 181
car.py, 162–179
cars.py, 43–45, 72
cities.py, 121 atributos
de clases, 159 acceder,
160 valores
predeterminados,
163 modificar, 164–166
crear, 158–162 importar,
174–179 todas las clases de
un módulo, 177 clases múltiples,
175–177 clase única, 174–
175
herencia, 167–173 atributos
y métodos, 169 clases secundarias,
167 método __init__() , 167–169
instancias como atributos, 170–172
métodos anulados, 170 clases
principales, 167 subclases, 168
función super() , 168 superclases,
168 instancias , 157 métodos, 159
llamada, 160 método __init__() , 159

modelado de objetos del mundo real, 173
instancias múltiples, 161 convenciones de
nomenclatura, 159 objetos, 157 pautas de
estilo, 181 archivos de valores separados
por comas. *Consulte* los archivos CSV
(valores separados por comas) *comment.py,* 29
comentarios, 29–30 pruebas
condicionales, 72–77. *Véase*
también sentencias if

confirm_users.py, 124
constantes, 28 *counting.py,*
118, 122 archivos CSV
(valores separados por comas),
334–346
comprobación de errores, 343–
345 análisis de encabezados, 334–
335 lectura de datos, 336

D

análisis de datos, 305
bases de datos. *Consulte* Django:
visualización de datos de bases de datos,
305. *Consulte también*
Matplotlib; Módulo de fecha y hora de
Plotly , 337–339 *death_valley_highs_lows.py,*
343–345 decoradores, 429 valores
predeterminados, 134 atributos de clase, 163
parámetros de función, 134 palabra clave
def , 130 instrucción del , 39 *dice_visual.py,*
328–330 definición de diccionarios, 92
diccionarios vacíos, 94 formato más grande,
97 método get() , 98 KeyError, 98 pares clave-
valor, 92–99 agregar, 93 eliminar, 96 recorrer,
99–105 claves, 101 claves en orden, 103
pares clave-valor, 99
valores, 104 ordenar, 94
ordenar una lista de, 374
valores
acceder, 93
modificar, 95
die.py, 324 *die_visual.py,*
325–327 *dimensiones.py,*
66–67 Discord, 48 div
(HTML), 441 división (/), 26
division_calculator.py, 194–
197 Django, 379 *Ver también*
Heroku ; Proyecto de registro de
aprendizaje
sitio de administración,
387–392 registrar modelos, 388, 391
asociar datos con usuarios, 435
Bootstrap, 438–447 tarjeta,
446 navegación
plegable, 440 elemento contenedor,
443 aplicación django-bootstrap4,
438
Encabezados HTML, 439–440
jumbotron, 443 barra de navegación,
440–442 formas de estilo, 444–
445 comandos
crear superusuario,
388 vaciar, 433 hacer
migraciones, 387, 391, 432
migrar, 383 shell, 392 startapp,
385, 421 startproject, 382 crear
un proyecto, 381 bases de datos
eliminación en cascada, 390
creación, 382 claves
foráneas, 390 relaciones de
varios a uno, 390 migración,
383, 391 consultas, 404,
433 conjuntos de consultas,
392–393, 401 campos
obligatorios (no anulables), 432

Django, bases de datos
continuas, *restablecimiento*
continuo , 433 SQLite,
383 decoradores, 429
implementación. *Consulte*
servidor de desarrollo Heroku,
383, 389 documentación Django, 379
modelos, 386 consultas, 394 plantillas,
406 formularios, 410–420 argumento
de acción , 413 falsificación de
solicitud entre sitios, 413 visualización,
413 solicitudes GET y POST, 412
ModelForm, 410, 414 relleno previo con
datos, 419 procesamiento, 412, 416
validación, 410 widgets, 414 método
get_object_or_404() , 460
hashes (para contraseñas), 388
elemento div HTML, 441 elemento
principal, 442 margen, 443 relleno, 443
elemento span, 442 HTTP 404 error, 434
INSTALLED_APPS, 386, 421, 438
instalando, 381 localhost, 383 cerrando
sesión, 424 página de inicio de sesión,
422 @login_required, 429 plantilla de inicio de
sesión, 422 URL de asignación, 395–396
migrando la base de datos, 383, 391 modelos,
385–393, 431 privilegios, 387 proyectos
(frente a aplicaciones), 384 función
redirect() , 411
página de registro, 426–428 ciclo de
lanzamiento, 381 restricción de acceso
a datos, 433–435 restricción de acceso
a páginas, 428–435
settings.py APLICACIONES
INSTALADAS, 386, 421, 438
LOGIN_URL, 429 SECRET_KEY, 461
shell, 392, 431 iniciar un app, 385
iniciar un nuevo proyecto, 382 archivos
estáticos, 448 estilo. *Consulte* Django:
superusuario de Bootstrap, 387 plantillas,
397 etiquetas de anclaje, 399 etiquetas de
bloque , 399 diccionario de contexto, 401
filtros, 405 for loop, 402 sangría en, 399
herencia, 398 filtro de saltos de línea , 405
etiquetas de plantilla, 399 aplicaciones de
terceros, 438 URL
capturar valores, 404 espacios
de nombres, 399
Patrones de URL, 395–396
etiqueta de plantilla de URL , 399
valores de ID de usuario, 431
usuarios
vista de inicio de sesión
predeterminada, 422 mostrar
mensaje al usuario que ha
iniciado sesión, 424 iniciar
sesión en un usuario, 427
UserCreationForm, 427 versiones,
381 vistas, 396 recuperar objetos,
401, 404 docstrings, 130 *dog.py,* 158–
162 notación de puntos, 151, 160 operador de
doble asterisco (**) , 149

Y

temblores. *Ver* mapeo de
terremotos
electric_car.py, 168–173
electric_car.py module, 178
función enumerate() , 335
variables de entorno, 456 epoch
time, 366 *eq_explore_data.py,*
348–351 operador de igualdad
(==), 72–73 *eq_world_map.py ,*
351–357 *even_numbers.py,* 58
even_or_odd.py, 117 excepciones,
183, 194–202

decidir qué errores informar,
201 bloques else ,
196 fallar silenciosamente,
200
FileNotFoundError, 197
manejo, 194 bloques try-
except , 194 uso para
evitar bloqueos, 195
ZeroDivisionError, 194
exponentes (**), 26

F

favoritos_idiomas.py, 97–98
FileNotFoundError, 197
file_reader.py, 184–188 cierre
de archivos, 185 rutas de

archivo, 185 absoluto, 186
relativo, 186 apertura, 184
modo agregar, 192
modo lectura, 192 modo
escritura, 192 lectura desde,
184 –190 archivos
completos, 184–185
línea por línea, 187
hacer una lista de líneas,
188 trabajar con
contenidos, 188 trabajar
con archivos grandes, 189

escribir
agregando, 193
archivos vacíos, 191
líneas múltiples, 192
first_numbers.py, 57
flags, 120 floats, 26
foods.py, 63–65 for
loops, 49. *Ver también*
diccionarios;

enumera *formatted_name.py,*
138–140 *full_name.py,* 21–22
funciones, 129 alias (as), 152
argumentos, 131–137
palabra clave arbitraria, 148
número arbitrario de, 147
evitar errores, 136 palabra
clave, 133 listas como, 143–
146 mezclar posicional y
arbitrario, 148 opcional, 138
orden de, 133 posicional,
132–133 incorporado,
471 llamadas, 130–137
llamadas equivalentes, 135–
136 varias veces, 132
definición, 130 diccionarios,
devolución, 140 listas en
modificación , 143–145
proteger, 145 módulos, 150–
155 alias (as), 153 importar todas
las funciones (*), 153 importar
módulos completos, 150 importar
funciones específicas, 152
parámetros, valores
predeterminados para, 134 pasar
información a. *Ver* funciones:
valores de retorno de
argumentos, 137–142
estilo, 154

GRAMO

juegos. *Ver* proyecto Alien Invasion;
Pygame Geany, 476
solicitudes GET, 412 obtener ayuda
Discord, 484 IRC (Internet Relay Chat),
482–483 documentación oficial de Python,
481
recursos en línea, 480 r/
learnpython, 482
depuración de patitos de goma,
480 Slack, 483 Stack Overflow, 481
tres preguntas principales, 479 Git,
360, 450 sucursales, 452, 488
confirmaciones, 360, 450 comprobación,
491–493 creación, 452 , 457, 460,
488, 490, 493 configurar, 451, 486
HEAD separado, 492 agregar
archivos, 452, 460, 488, 493
ignorar, 451, 487 HEAD, 492
instalar, 450, 486 registrar, verificar,
489 repositorios, 360 eliminar, 493
inicializar, 452, 487, 493 revertir
cambios, 490 estado, verificar,
452, 457, 487–493 GitHub, 360
mayor que (>), 75 mayor que o igual
a (>=), 75 *greetinger.py,* 114, 130–131
greeting_users.py, 143 paquete
gunicorn , 448

H

Hacker News, 372
hash mark (#), para comentarios, 29
head, de un archivo HTML, 440 HEAD
(Git), 492 Hello World, 9 *hello_git.py,*
486–491 *hello_world.py,* 10, 15–19
Heroku, 437. *Véase también* Django;
Git; Proyecto de registro de aprendizaje
Bash shell, 454 CLI, instalación, 458
configuración de comandos,
458 destruir, 462 iniciar sesión, 453
abrir, 453 ps, 453 renombrar, 455
ejecutar, 454 establecer, 458 bases
de datos, configurar, 454 paquete
django-heroku , 448
documentación , 453 variables
de entorno, configuración, 456–
458 páginas de error,
personalizado, 458–460 plan
gratuito, limitaciones de, 448,
456 creación de una cuenta, 448
Procfile, 450 eliminación de
proyectos, 461 envío a Heroku, 452–
453, 457 visualización en directo ,
453 Tiempo de ejecución de Python,
especificación, 449
requisitos.txt, 448–449 protección,
456 *settings.py,* modificación para,
450, 456, 459 superusuario,
creación, 454–455 URL, fácil de usar,
455 archivos ocultos, 451

hn_article.py, 372
hn_submissions.py, 373
Elaboración casera, 469

i

IDE (entorno de desarrollo integrado),
473
INACTIVO,
475 sentencias if
y palabra clave, 75
expresiones booleanas, 77
comprobación de igualdad (==),
72 igualdad, caso omiso, 73
listas vacías, 87 desigualdad (!=),
74 elementos en una lista, 76
elementos que no están en una
lista, 77 pruebas condicionales, 72–
77 declaración elif , 80–84
declaración else , 79 listas y, 85–88
comparaciones numéricas, 74–76 or
palabra clave, 76 simple, 78 pautas de
estilo, 90 probar múltiples condiciones,
83–84
inmutable, 65
importar *, 153
importar esto, 30
errores de sangría, 53–56 errores
de índice, 47 operador de
desigualdad (!=), 74 bucles infinitos,
122 herencia, 167. *Ver también*
clases: herencia función input() , 114–117
entrada numérica, 115–116
indicaciones, 114 método insert() ,
38
IRC (Internet Relay Chat), 482–483 función
itemgetter() , 374 método items() , 100

j

Archivos JSON
que examinan datos, 347
formato de archivo geoJSON, 349
función json.dump() , 348 función
json.load() , 204 jumbotron, 443

k

método keys() , 101
pares clave-valor, 92–99. *Véase también*
argumentos de palabras
clave de diccionarios, 132, 133. *Véase también*
palabras clave de funciones,
471

L

language_survey.py, 217
Proyecto de registro de aprendizaje, 379. *Véase*
también Django; Despliegue de
Heroku. *Consulte* Git, Heroku files *404.html,*
458 *500.html,* 459 *admin.py,* 388 *base.html,*
399, 403, 423, 425, 428, 439–443
edit_entry.html, 419 *forms.py,* 410,
414 . *gitignore,* 451 *index.html,* 397,
443 *learning_logs/ urls.py,* 395, 401,
404, 411, 415, 418 *learning_log/*
urls.py, 395, 422 *login_out.html,* 425
login.html, 422–423, 444 *models.py,*
385, 390, 431 *new_entry.html,* 416
new_topic.html, 412 *Procfile,* 450
register.html, 427 *requirements.txt,*
448 *runtime.txt,* 449 *settings.py,*
386, 421, 429, 450, 456 , 459

Proyecto de registro de aprendizaje,
archivos continuados, *topic.html*
continuo, 404–405, 417, 420
topics.html, 402, 405, 413, 445
users/ urls.py, 422, 426 *users/*
views.py, 426 *views.py,* 396, 401, 404,
411, 415, 418, 429, 430,
433, 460
páginas
editar entrada, 418–420
inicio, 394–398 iniciar
sesión, 422–424 cerrar
sesión, 424 nueva
entrada, 414–417 nuevo
tema, 410–413 registro,
426 tema, 403 temas,
400–403 aplicación de
usuarios , 421–428
virtual entorno, 380–381
escribir una especificación (spec),
380 función len() , 45 menor que (<),
75 menor o igual que (<=),
75

linux
Hola mundo, corriendo, 10
Python
comprobando la versión instalada,
8 instalación, 470 solución de
problemas de instalación, 11

Sublime Text, instalación,
terminal 9
ejecutando programas desde, 12
iniciando una sesión de Python, 9
listas, 33 como argumentos, 143–146
copiando, 63–65 elementos accediendo,
34 accediendo al último, 35 agregando
con append(), 37 agregando con
insert(), 38 identificando único, 104
modificando, 36
eliminando con del, 39
eliminando con pop(), 39–41
eliminando con remove(), 41
vacío, 38 función enumerate() , 335
bucles for , 49–56 anidados, 109, 263
sentencias if y, 85–88 errores de
sangría , 53–56 índices, 35
errores, 46–47 negativo, 35 función
len() , 45 listas de comprensión, 59
denominación, 34 listas numéricas,
57–60 función range() , 58–59
eliminación de todas las
apariciones de un valor, 125 divisiones,
61–63 clasificación, 43–46 método
reverse() , 45 función sorted() , 44
método sort() , 43 errores lógicos, 54

METRO

Mac OS
Hola mundo, corriendo, 10
Python
comprobando la versión instalada,
7 instalando, con Homebrew, 469–470
instalación, versión
oficial, 7–8
solución de problemas
de instalación, 11
Sublime Text, instalación, 8
terminales
ejecutando programas desde, 12
iniciando una sesión de Python, 8
magicians.py, 50–53 *magic_number.py,* 75
making_pizzas.py, 151–153 mapeando
terremotos, 347–357 construyendo un
mapa del mundo, 351 escalas de colores,
354

descarga de datos, 347, 358
examen de datos JSON, 347
extracción de ubicaciones, 351
extracción de magnitudes, 350
formato de archivo geoJSON, 349
texto flotante, 356 ordenación de latitud y
longitud, 349 magnitudes, representación,
353
Tipo de gráfico Scattergeo, 352
Matplotlib, 306–323, 336–347
hachas
método axis() , 313
variable ax , 307
eliminación, 321 variable
fig , 307 diagramas de formato
argumento alfa, 342 estilos
incorporados, 310 mapas de
color, 314 colores
personalizados, 314
etiquetas, 307–308 grosor
de línea, 307 sombreado,
342 tamaño, 322 galería,
306 instalación, 306 gráficos
de líneas, 306 método plot() ,
307, 309 módulo pyplot , 307
guardar gráficos, 315 gráficos de
dispersión, 310–313 función
subplots () , 307 métodos, 20,
236–237. *Ver también* clases

módulos, 150. *Ver también* clases:
módulos; funciones:
módulos
operador módulo (%), 116–117, 122
motocicletas.py, 37–42 *mountain_poll.py,* 126
mpl_squares.py, 306–310 multiplicación (*), 26
my_car.py, 175 *my_cars.py,* 177–179
my_electric_car. pi, 176

norte

errores de nombre,
17 *función_nombre.py,* 210–215
nombre.py, 20 *nombres.py,* 210
anidamiento, 106–112 profundidad,
110 diccionarios en diccionarios,
110–111 diccionarios en listas,
106–108 listas en diccionarios, 108–
110 nueva línea (\n), 22
Ninguno, 99 no (!), 74 *number_reader.py,*
203 números, 25–28 aritmética, 26
comparaciones, 74–76 exponentes, 26
flotantes, 26 división de piso (//), 260
enteros, 26 mezclar enteros y flotantes, 27
orden de operaciones, 26 función round() ,
294 guiones bajos en, 28 *number_writer.py,*
203

LA

programación orientada a objetos, 157.
Ver también clases
función open() , 184 o
palabra clave, 76

PAG

parámetros, 131
clases de padres, 167. *Consulte también*
clases: herencia *loro.py,* 114,
118–121 declaración de paso , 200 PEP 8, 68–
70, 90, 154 *persona.py,* 140–142 *mascotas.py,*
125, 132 –136

pip, 228
instalando Django, 381
instalando Matplotlib, 306
instalando Plotly, 324
instalando Pygame, 228
instalando Requests, 361
pi_string.py, 188–190 *pizza.py,*
147–148 *players.py,* 61–63 Plotly,
306 Bar() clase, 327 datos,
formato largo, 353 barras de
gráficos de formato, 368
escalas de colores, 354 texto
flotante, 356, 369–370 color
de marcador, 368 tamaño
de marcador, 353 Clase
Layout() , 327 diseño,
formato largo, 367 eje x, 368
eje y, 368 galería, 324
histograma, 326 instalación,
324 función offline.plot() ,
327 referencia de figura de
Python, 371 tipo de gráfico
Scattergeo, 352 guía del usuario,
371 argumentos posicionales.
Ver funciones: argumentos
solicitudes POST, 412
printing_models.py, 143–146
Project Gutenberg, 198 extensión
de archivo .py , 16 Pygame.
Consulte también colores de fondo del
proyecto Alien Invasion, 230–
231 colisiones, 268–269, 272–275, 291–
292 colores, 231 crear ventanas vacías,
229 cursor, ocultar, 285 mostrar texto,
280 finalizar juegos, 276 bucles de
eventos, 230
modo de pantalla
completa, 244 grupos,
248 dibujar todos los elementos,
259 vaciar, 271 recorrer, 250
eliminar elementos de, 251
almacenar elementos, 248
actualizar todos los elementos,
249 imágenes, dibujar en pantalla,
235 imágenes, cargar, 234 instalar,
228 print() llama, 251 salir, 244 rect
objetos creando desde cero, 247
posicionamiento, 234, 247–248, 257–
258, 260–261, 263–264 respondiendo
a la entrada, 230 pulsaciones de
teclas, 238 clics del mouse, 283
pantalla coordenadas, 234
superficies, 230 juegos de
prueba, 270 módulo pyplot ,
307

Python
>>> indicador,
4 funciones integradas,
471 documentación, 481
instalación en Linux, 470
en macOS, oficial, 7–
8 en macOS, usando
Homebrew, 469–470
en Windows, 5–6, 467–469
intérprete, 16 palabras clave, 471
PEP 8, 68–70, 90, 154 biblioteca
estándar, 180–181 sesiones de
terminal, 4 versiones, 4 Zen of, 30–
31 Python Enhancement Proposal
(PEP), 68 *python_repos.py,* 361–
365 *python_repos_visual.py,* 366–
371

q

valores para dejar de fumar, 118–120

R

random_walk.py, 316–317
paseos aleatorios, 315–323
función choice() , 317 puntos
de color, 319 método
fill_walk() , 316 paseos
múltiples, generación, 318 trazado, 317
Clase RandomWalk , 316 puntos de inicio
y fin, 320 rango( ) función, 57–59 método
readlines() , 188 método read() , 185
refactorización, 206–208, 236 *recordar_me.py,*
204–208 Paquete de solicitudes, 361 valores
devueltos, 137 r/learnpython, 482
rollercoaster.py, 116 tirando dados, 323–330
analizando resultados, 325 Clase de dado ,
324 tamaños diferentes, 329 función randint() ,
324 dos dados, 328 depuración de pato de
goma, 480 *rw_visual.py,* 317–323

S

scatter_squares.py, 311–315
sets, 104 *sitka_highs_lows.py,*
340–342 *sitka_highs.py,* 334–340
Slack, 483 función sleep() , 274 slice,
61 método split() , 198 corchetes ([]),
34 *squares.py,* 58, 60 Stack Overflow,
481 almacenamiento de datos, 202–
205. *Consulte también* cadenas
JSON, 19–25 cambio de mayúsculas
y minúsculas, 20 método format() ,
22
cadenas f, 21
nuevas líneas, 22
comillas simples y dobles, 19, 24
tabulaciones, 22 uso de variables, 21
espacio en blanco, 22–24 método
strptime() , 338 pautas de estilo, 68–70
líneas en blanco, 69
CamelCase, 181
clases, 181
funciones, 154
sentencias if , 90
sangría, 69 longitud
de línea, 69
PEP 8, 68
Texto sublime, 4–10, 474–475
comentar el código, 475 configurar,
9 personalizar, 474 sangrar y quitar
sangría de bloques de código, 474
instalar, 7–9 indicador de longitud
de línea, 474 ejecutar
programas de Python, 9–10 guardar
su configuración, 475 tabuladores y
espacios, 474 restar (-), 26 superclases,
168. *Ver también* clases: herencia
encuesta.py, 217 errores de sintaxis, 24
resaltado de sintaxis, 16

T

tabulador (\t),
22 código de prueba, 209–
222 agregar pruebas,
214 métodos de afirmación,
216 cobertura, 211
pruebas fallidas, 212–214
pasar pruebas, 211–212
método setUp() , 220 caso
de prueba, 211 clases de
prueba, 216 –221 funciones
de prueba, 210–215 módulo de
prueba unitaria, 209 pruebas
unitarias, 211

test_name_function.py, 211–215
test_survey.py, 218–221 editores de texto e IDE Atom, 476 Emacs y Vim,
476 Geany, 476 IDLE, 475
Jupyter Notebooks, 477
PyCharm, 476 Sublime Text, 4–10, 474–475 Visual Studio Code,
476 *toppings.py,* 74, 83–88
rastreo, 18 bloques de prueba
excepto , 194–202. *Véase también*
tuplas de excepciones, 65–67
errores de tipo, 66

EN

módulo unittest , 209
pruebas unitarias, 211
tiempo Unix, 366
user_profile.py, 148

EN

método de valores() , 104
variables, 16–19, 28
constantes, 28 como
etiquetas, 18
asignación múltiple, 28
convenciones de nomenclatura,
17 valores, 16 sistema de
control de versiones, 485. *Consulte*
también el entorno virtual de Git (venv),
380 *vote.py,* 79–80

En

datos meteorológicos, 334–
347 marco web, 379 bucles
while , 118–127 bandera
activa , 120–121
instrucción break , 121
infinito, 122 mover
elementos entre listas, 124 valores de
salida, 118–120 espacios en blanco,
22–24
ventanas
Hola mundo, corriendo, 10
Python
comprobación de la versión
instalada, 5 instalación, 5–6, 467–
469 solución de problemas de
instalación, 11, 467–469
Sublime Text, instalando, 7
terminales
ejecutar programas desde, 12
iniciar una sesión de Python, 6 con
declaración, 185 *word_count.py,* 199–201
write_message.py, 191–193 método
write() , 192

DESDE

Zen de Python, 30–31
Error de división cero, 194



Python Crash Course, 2nd Edition está ambientado en New Baskerville, Futura, Dogma y The Sans Mono Condensed.

RECURSOS

Visite *https:// nostarch.com/ pythoncrashcourse2e/* para obtener recursos, erratas y más información.

Más libros serios de  Prensa sin almidón



Machine Translated by Google

MUNDIAL

MEJOR VENDIDO

MÁS DE 500.000

COPIAS VENDIDAS



APRENDER PITÓN —

¡RÁPIDO!

Python Crash Course es la guía más vendida del mundo sobre el lenguaje de programación Python. Esta introducción rápida y completa a la programación con Python te permitirá escribir programas, resolver problemas y crear cosas que funcionen en poco tiempo.

En la primera mitad del libro, aprenderá conceptos básicos de programación, como variables, listas, clases y bucles, y practicará la escritura de código limpio con ejercicios para cada tema. También aprenderá cómo hacer que sus programas sean interactivos y probar su código de manera segura antes de agregarlo a un proyecto. En la segunda mitad, pondrá en práctica sus nuevos conocimientos con tres proyectos sustanciales: un juego de arcade inspirado en Space Invaders, un conjunto de visualizaciones de datos con las prácticas bibliotecas de Python y una aplicación web simple que puede implementar en línea.

A medida que avance en el libro, aprenderá a:

- Utilice potentes bibliotecas y herramientas de Python, incluidas Pygame, Matplotlib, Plotly y Django
- Cree juegos en 2D que respondan a las pulsaciones de teclas y clics del mouse, y que aumenten en dificultad
- Usar datos para generar visualizaciones interactivas
- Cree y personalice aplicaciones web e impleméntelas de forma segura en línea
- Tratar errores y errores para que pueda resolver su problemas de programación propios

Esta **segunda edición actualizada** se revisó minuciosamente para reflejar lo último en código y prácticas de Python. La primera mitad del libro incluye una cobertura mejorada de temas como cadenas f, constantes y administración de datos. En la segunda mitad, el código de los proyectos se actualizó con una mejor estructura, una sintaxis más limpia y bibliotecas y herramientas más populares y actualizadas, como Plotly y la última versión de Django. (Para obtener una lista completa de las actualizaciones, consulte el Prefacio).

Si ha estado pensando en profundizar en la programación, Python Crash Course lo ayudará a escribir programas reales rápidamente. ¿Por qué esperar más? ¡Arranque sus motores y código!

SOBRE EL AUTOR

Eric Matthes es un maestro de ciencias, matemáticas y programación de secundaria que vive en Alaska. Ha estado escribiendo programas desde que tenía cinco años y es el autor de Python Flash Cards, también de No Starch Press.

CUBIERTAS PITÓN 3.X



LO MEJOR EN GE EKE NTERTA INME NT ™
www.nostarch.com
$39.95 ($53.95 CDN)